Octavian CUIBUS

Tiberiu LEȚIA

SISTEME EVOLUTIVE

Îndrumător de laborator

UTPRESS
Cluj-Napoca, 2020
ISBN 978-606-737-485-8

Octavian CUIBUS **Tiberiu LEŢIA**

SISTEME EVOLUTIVE

Îndrumător de laborator

UTPRESS
Cluj - Napoca, 2020
ISBN 978-606-737-485-8

Editura U.T.PRESS
Str. Observatorului nr. 34
C.P. 42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
http://biblioteca.utcluj.ro/editura

Director: ing. Călin Câmpean

Recenzia: Prof.dr.ing. Adina Aştilean
Ş.l.dr.ing. Mihai Hulea

ISBN 978-606-737-485-8
Bun de tipar: 05.12.2020

Cuprins

Laborator 1

Introducere în sisteme evolutive – algoritmi genetici

1. Regulament de disciplină

- Prezența la orele de laborator este obligatorie.
- Studenții au obligația de a respecta orarul laboratorului și grupele/semigrupele din care fac parte
- Recuperarea orelor de laborator se face în conformitate cu regulamentul universității.
- Fiecare student va fi evaluat în urma activităților de laborator, nota minimă cu care studentul se consideră promovat fiind 5. Studenții care au la sfârșitul laboratorului situația neîncheiată sau încheiată cu nota sub 5 nu vor putea susține examenul de disciplină.

2. Obiectivele laboratorului

- Prezentarea regulamentului de disciplină
- Prezentarea uneltelor necesare pentru orele de laborator (JGAP / AForge.Net)
- Însușirea modului de lucru pentru soluționarea unor probleme simple folosind algoritmi genetici
- Evaluarea execuției algoritmului genetic: convergență, timpul de execuție, eficiența găsirii soluției, etc.

3. Prezentare teoretică a algoritmilor genetici

Algoritmii genetici fac parte din clasa algoritmilor evolutivi, care vizează căutarea soluției unei probleme, în mod iterativ, prin optimizarea la fiecare pas a unui indice de performanță asociat soluției.

Cu scop recapitulativ, se oferă în Figura 1. schema logică a algoritmilor genetici [1], [2]. Modul cel mai comun de abordare implică următorii pași:

1. Stabilirea structurii cromozomului: numărul de gene și genotipul fiecăreia
2. Generarea populației inițiale
3. Construirea indivizilor din cromozomi cu ajutorul funcției de *mapping*
4. Evaluarea indivizilor folosind funcția de *fitness*
5. Selecția indivizilor
6. Crearea descendenților prin încrucișare (engl. *crossover*), mutație, etc.
7. Stabilirea populației care constituie generația următoare

După faza de inițializare, pașii 3-7 se efectuează într-o buclă repetitivă pentru un anumit număr de generații sau până când soluția găsită îndeplinește anumite criterii de performanță.

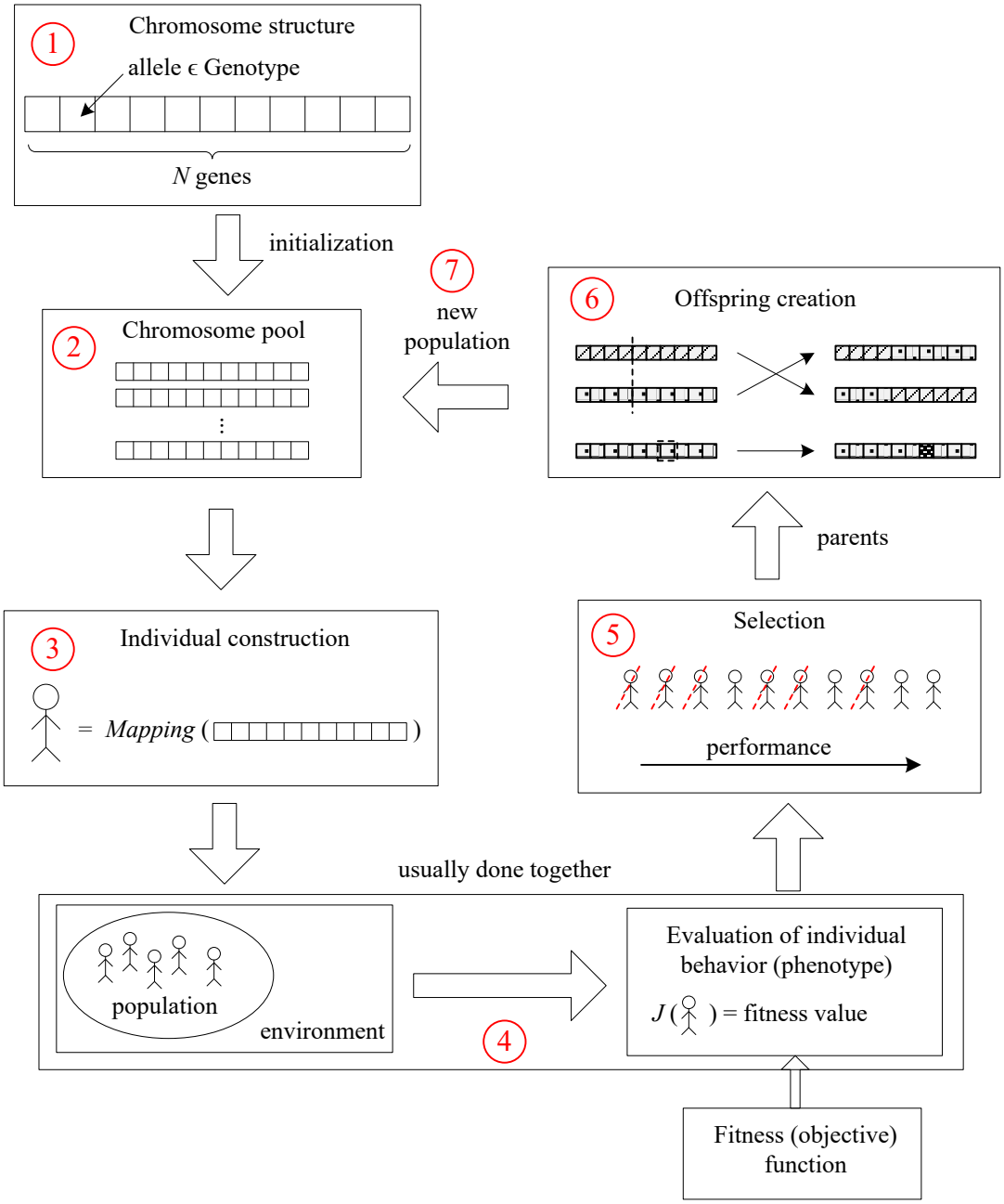


Figura 1. Schema logică a algoritmilor genetici

4. Unelte necesare pentru laborator

În majoritatea limbajelor de programare există biblioteci sau pachete care implementează o mare parte a funcționalităților la abordarea cu algoritmi genetici (generarea populației inițiale, modalitățile de selecție, operatorii de crossover și mutație), lăsând în grija programatorului doar structura cromozomului și funcțiile de mapping și de performanță. Dintre cele mai folosite amintim:

- pentru Java SE: JGAP (Java Genetic Algorithm Package), Jenetics, etc

- pentru C#: Aforge.Net, etc
- pentru Matlab: Genetic Algorithm Toolbox
- pentru Python: pygalib
- pentru Javascript: genetic-js

Pentru obiectivul acestui laborator se va folosi varianta Java SE, pachetul JGAP [3], pentru că pune la dispoziția programatorului mai multe exemple de programe care pot fi rulate direct, sau modificate relativ facil pentru a rezolva alte probleme. De asemenea, este open source, deci putem adapta de exemplu, operatorii de crossover și mutație la necesitățile noastre.

Este recomandat, dar nu obligatoriu, să se folosească mediul de dezvoltare Eclipse. Se va descărca versiunea *Eclipse IDE for Java Developers* de pe pagina http://www.eclipse.org/downloads/index.php. Se dezarhivează în locația dorită și se execută programul *eclipse.exe*. Este necesar să existe preinstalat și pachetul *JDK* (*java development kit*), de exemplu varianta *Java 2 SE 1.7*. Când se descarcă pachetul *JDK* trebuie avut grijă să se selecteze platforma și sistemul de operare corespunzătoare.

Se va descărca de la JGAP repository (https://sourceforge.net/projects/jgap/) ultima versiune a bibliotecii (de la secțiunea *Files*). În cadrul acestui laborator se va lucra pe exemplele distribuite în varianta *jgap_3.6.3_full*. Fiind practic o bibliotecă, pachetul JGAP nu necesită instalare, ci se dezarhivează fișierul descărcat și se referențiază ca și bibliotecă în Eclipse (pentru aceasta, click dreapta pe numele proiectului, se selectează opțiunea *Properties*, apoi *Java Build Path* din meniul din stânga, tabul *Libraries*, click pe *Add External JARs…*, apoi se selectează locația unde a fost dezarhivat pachetul JGAP și se selectează fișierul *jgap.jar*).

În figura 4 se prezintă diagrama UML a unor elemente esențiale din pachetul JGAP. Se observă că majoritatea funcționalităților sunt implementate deja în clasele existente. Contribuția programatorului este indicată cu contur roșu:

- definirea funcției de performanță (prin extinderea clasei `FitnessFunction` cu o clasă proprie care să implementeze metoda `evaluate()`)
- în programul principal, crearea obiectului de configurare, a populației inițiale, și apoi apelarea metodei `evolve()` din clasa `Genotype`.

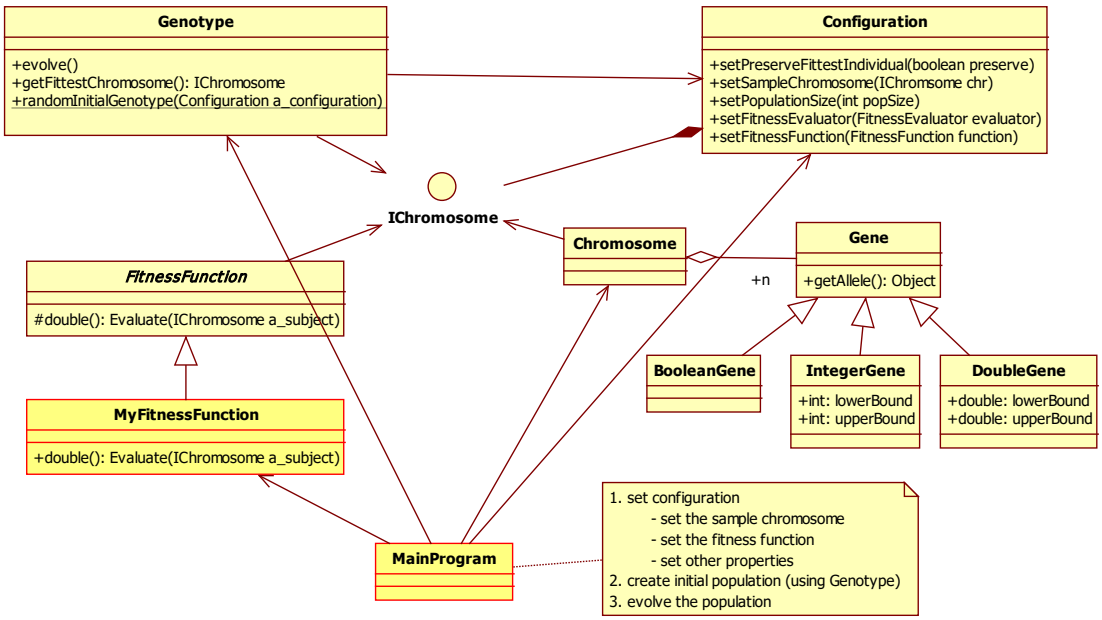


Figura 2. Structura de bază a pachetului JGAP

5. Problemă rezolvată - prezentarea modului de lucru

5.1. Formularea problemei

O problemă simplă care poate fi rezolvată cu algoritmi genetici este aceea de a calcula restul în monede, astfel încât suma să fie cât mai aproape de valoarea dorită, folosind cât mai puține monede. Aici, o soluție performantă va reprezenta un compromis între cele două cerințe. Există 4 tipuri de monede: Quarters (valoarea 0.25), Dimes (0.1), Nickels (0.05), Pennies (0.01). Sunt disponibile în total 20 de monede de tip Quarter, 30 Dimes, 50 Nickels, respectiv 80 Pennies.

5.2. Rezolvarea problemei

Necunoscuta problemei reprezintă informația despre numărul de monede din fiecare tip. Soluția problemei va fi deci un vector de 4 numere întregi. Această soluție trebuie codificată în cromozom, astfel încât să se acopere tot spațiul de căutare.

Prin urmare, vom considera un cromozom format din 4 gene cu alele numere întregi, iar valoarea alelei fiecărei gene va reprezenta câte monede din acel tip vor fi considerate. Genotipul fiecărei gene este o mulțime de numere întregi, diferită pentru fiecare genă. De exemplu, $G_1 = \{0, 1, 2, ..., 20\}$, pentru că sunt maxim 20 de monede de tip Quarter disponibile.

Algoritmul genetic va genera soluții care trebuie evaluate în funcție de obiectivele problemei, deci conform cu o funcție de performanță. Pentru evaluarea unei soluții posibile, vom evalua cele două obiective: suma sa fie cât mai aproape de suma dorită (S) și numărul total de monede să fie cât mai mic. Expresia funcției de performanță este deci:

$$J = \alpha \cdot \left| S - \sum_{i=1}^{4} g_i \cdot V_i \right|^2 + \sum_{i=1}^{4} g_i ,$$

unde g_i reprezintă valoarea genei i (numărul de monedede tipul i), V_i este valoarea modei de tipul i, iar α este un factor de scalare. Cele două contribuții la funcția de performanță sunt însumate, performanța fiind cu atât mai bună cu cât J are valoare mai mică.

Pentru a stabili cât anume contează fiecare obiectiv la funcția globală de performanță se pot folosi diverse artificii:

- un factor liniar de scalare α pentru a stabili prioritatea fiecărei contribuții
- penalizarea progresivă a erorii folosind ridicarea la pătrat. Astfel, o eroare mică va fi penalizată puțin, iar o eroare mare va fi penalizată puternic.
- penalizarea diferențiată în funcție de diferite condiții logice, etc.

5.3. Implementare

În continuare vom parcurge codul sursă al programului *ConstraintExample* (*JGAP/examples/src/examples/constraint*). Există două fișiere:

- *ConstraintExample.java* – conține programul principal unde se setează toți parametrii necesari pentru a rula programul. Aici se află și bucla iterativă descrisă în capitolul 3.
- *SampleFitnessFunction*.java – conține implementarea funcției de performanță (funcția obiectiv)

Redăm mai jos secvența de cod simplificată a programului principal, unde am eliminat toate detaliile care nu sunt absolut necesare pentru rularea programului:

Secvența de cod 1: Exemplu simplificat

```
import org.jgap.*;
import org.jgap.impl.*;
public class ConstraintExample {
  private static final int NUM_EVOLUTIONS = 100;
  public static void main(String[] args) throws
InvalidConfigurationException{
    double targetAmount = 1.84;
    Configuration conf = new DefaultConfiguration();
     Configuration.resetProperty(Configuration.PROPERTY_FITEVAL_INST);
     //we use Delta evaluator (low value for fitness = better):
     conf.setFitnessEvaluator(new DeltaFitnessEvaluator());
     conf.setPreservFittestIndividual(true);
     conf.setKeepPopulationSizeConstant(true);

     FitnessFunction fitnessFunction = new
SampleFitnessFunction(targetAmount);
     conf.setFitnessFunction(fitnessFunction);

     Gene[] sampleGenes = new Gene[4];
     sampleGenes[0] = new IntegerGene(conf, 0, 20); // Quarters
     sampleGenes[1] = new IntegerGene(conf, 0, 30); // Dimes
     sampleGenes[2] = new IntegerGene(conf, 0, 50); // Nickels
     sampleGenes[3] = new IntegerGene(conf, 0, 80); // Pennies
     IChromosome sampleChromosome = new Chromosome(conf, sampleGenes);
     conf.setSampleChromosome(sampleChromosome);
```

```
     conf.setPopulationSize(80);
     Genotype population = Genotype.randomInitialGenotype(conf);

     for (int i = 0; i < NUM_EVOLUTIONS; i++) {
       population.evolve();
       IChromosome bestSolutionSoFar =
population.getFittestChromosome();
       DisplayIndividual(bestSolutionSoFar);
     }
   }

   public static void DisplayIndividual(IChromosome chr){
     System.out.print("Fitness value: " + chr.getFitnessValue());
     System.out.print(", Coins: ");
     for (int i = 0; i < 4; i++)
       System.out.print(SampleFitnessFunction.getNrCoinsAtGene(chr, i)
+ " ");
     System.out.println(", total change: " +
SampleFitnessFunction.amountOfChange(chr));
   }
}
```

Mai întâi se construiește obiectul de configurare `gaConf` în varianta `DefaultConfiguration`, care va conține toți parametrii necesari rulării algoritmilor genetici:

`DeltaFitnessEvaluator()` – se specifică faptul că o valoare numerică mică returnată de funcția de evaluare reprezintă o performanță mare a individului. Această setare a evaluatorului se folosește atunci când avem nevoie să minimizăm o anumită cantitate – în acest caz, diferența între obiectiv și valoarea curentă. Pentru a putea seta un nou `fitnessEvaluator` trebuie în prealabil resetată valoarea `PROPERTY_FITEVAL_INST`.

`setPreservFittestIndividual()` – se optează pentru păstrarea întotdeauna a celui mai bun individ (*ellite selection* cu un singur individ in elită)

`setKeepPopulationSizeConstant()` – mărimea populației este menținută constantă în fiecare generație, având valoarea setată cu instrucțiunea `gaConf.setPopulationSize(80)`

`setFitnessFunction()` – se specifică o instanță a clasei ce reprezintă funcția de fitness cu care se evaluează fiecare individ. Această clasă extinde `FitnessFunction` și suprascrie metoda `evaluate()`, care este folosită la evaluarea performanței.

`setSampleChromosome()` – structura cromozomului este setată folosind un obiect de tip `IChromosome`, pe baza căruia programul va genera cromozomii aleatori din populația inițială. În aces caz, cromozomul are 4 gene de tip `IntegerGene`, fiecare având setate valori minime și maxime între care se află valoarea alelei.

`randomInitialGenotype()` – se creează populația inițială aleatoare

În secvența de cod 1, bucla iterativă va executa un număr fix de 100 de iterații (`NUM_EVOLUTIONS`). Fiecare iterație reprezintă o întreagă generație a evoluției (mapping, evaluare, selecție, crearea descendenților), practic pașii 3-7 descriși în capitolul 3. Metoda care implementează ”trecerea” unei generații este `genotype.evolve()`. De

asemenea, se tipărește pe ecran valoarea funcției obiectiv pentru cel mai performant cromozom din fiecare generație și valorile alelelor cromozomului.

În secvența de cod 2 se prezintă funcția de evaluare, care în metoda `evaluate(…)` calculează performanța unui cromozom dat. În primul rând, se calculează valoarea reprezentată de cromozom (folosind funcția de mapping), iar apoi, conform cu cerința problemei, se penalizează diferența până la valoarea obiectiv `targetAmount`, precum și numărul de monede. Se folosește pătratul valorii `changeDifference` pentru a introduce penalizări progresive (dacă valoarea este foarte departe de obiectiv, se penalizează puternic, altfel mai puțin). De asemenea, există un factor de scalare (`300`), care specifică prioritatea mai mare a valorii în fața numărului de monede.

Secvența de cod 2: Funcția de performanță

```
package lab1;

import org.jgap.*;
public class SampleFitnessFunction extends FitnessFunction {
    private final double targetAmount;

    public SampleFitnessFunction(double targetAmount) {
     this.targetAmount = targetAmount;
    }

    public double evaluate(IChromosome chr) {
     double changeAmount = amountOfChange(chr); //mapping
     int totalCoins = getTotalNumberOfCoins(chr); //mapping

     double changeDifference = Math.abs(targetAmount - changeAmount);
     double fitness = 300 * changeDifference * changeDifference;
     fitness += totalCoins > 1 ? totalCoins : 0;
     return fitness;
    }

    public static double amountOfChange(IChromosome chr) {
      int numQuarters = getNrCoinsAtGene(chr, 0);
      int numDimes = getNrCoinsAtGene(chr, 1);
      int numNickels = getNrCoinsAtGene(chr, 2);
      int numPennies = getNrCoinsAtGene(chr, 3);
      return (numQuarters * 0.25) + (numDimes * 0.1) + (numNickels *
0.05) + (numPennies*0.01);
    }

    public static int getNrCoinsAtGene(IChromosome chr, int position) {
      Integer numCoins = (Integer)chr.getGene(position).getAllele();
      return numCoins.intValue();
    }

    public static int getTotalNumberOfCoins(IChromosome chr) {
      int totalCoins = 0;
      for (int i = 0; i < chr.size(); i++)
        totalCoins += getNrCoinsAtGene(chr, i);

      return totalCoins;
    }
  }
```

5.4. Testarea programului și măsurarea performanțelor

Convergența algoritmului

Atunci când se rulează programul de mai sus se poate observa că performanța afișată crește la fiecare pas (deci valoarea funcției de evaluare scade). Acest fapt este asigurat de păstrarea celui mai bun individ din fiecare generație prin apelul metodei `setPreservFittestIndividual()`. Faptul că la fiecare pas soluția poate fi doar îmbunătățită conferă, în acest caz, convergența algoritmului [4] (în unele cazuri, este posibil ca soluția găsită să fie doar un optim local, nu global, acest caz fiind analizat în lucrările următoare). Este posibil ca, în timpul rulării programului, abaterea de la suma totală să crească de la o generație la alta. Aceasta se întâmplă numai în cazul în care performanța totală se îmbunătățește datorită scăderii numărului de monede.

Caracterul aleator al algoritmilor genetici este demonstrat de faptul că fiecare rulare produce, de cele mai multe ori, rezultate diferite. Caracterul aleator este dat de: generarea populației inițiale, operatorii de încrucișare și mutație și selecția indivizilor.

Eficiența algoritmului

Pentru a măsura timpul de execuție al algoritmului, se pot introduce niște instrucțiuni suplimentare. Timpul de execuție depinde, evident, de mașina pe care rulează programul, deci este mai relevant să analizăm eficiența algoritmului.

Eficiența algoritmului este dată de numărul de evaluări ale funcției de performanță raportat la mărimea spațiului de căutare. Spațiul de căutare este $S=G_1 \times G_2 \times G_3 \times G_4$, unde G_i este genotipul pentru gena i. Pentru problema dată, spațiul de căutare conține $|S|=20 \cdot 30 \cdot 50 \cdot 80 = 2\ 400\ 000$ soluții posibile. Evaluarea performanței se face pentru fiecare cromozom din fiecare generație, deci numărul de evaluări se poate calcula înmulțind numărul de generații cu mărimea populației (în acest caz 100 generații ·80 indivizi = 8000 de evaluări). Putem deci concluziona că s-au evaluat 8000 / 2400000 = 0.33% dintre soluțiile posibile.

De fapt, la rularea programului se observă că după o anumită generație soluția nu se mai îmbunătățește deloc. Întrucât fiecare rulare produce rezultate diferite, putem considera o medie a numărului de generații în care este găsită soluția finală și calculăm eficiența algoritmului cu noua valoare.

6. Problemă propusă

În continuare se va aborda o problemă clasică de optimizare, numită în literatură problema rucsacului (engl. *knapsack problem*). Se dă un rucsac cu volum dat V și n obiecte care trebuie introduse în rucsac volumul o_1, o_2, ..., o_n. Fiecare obiect are un anumit volum și o valoare asociată. Fără a depăși rucsacului, se cere să se maximizeze valoarea totală a obiectelor în rucsac.

Pentru un caz particular, se consideră n=10, V=30, iar volumul și valoarea fiecărui obiect sunt date de: $vol_i = i$, $val_i = i^2$, unde i=1, 2, ..., n.

Modul de abordare prezentat anterior este potrivit și pentru această problemă. Soluția problemei conține informația despre fiecare obiect dacă este considerat sau nu, prin urmare cromozomul va fi format n gene de tip `BooleanGene`. Pentru a calcula performanța J a unui cromozom, vom calcula valoarea totală a obiectelor și o penalizare

diferențiată J_{pen}: dacă volumul rucsacului este depășit se va penaliza performanța cu o valoare foarte mare și progresivă în funcție de cât este depășirea:

$$J = \sum_{i=1}^{4} g_i \cdot val_i + J_{pen}, \text{ unde } J_{pen} = if\left(V > V_{tot}\right) then\ 0\ else\ \alpha \cdot \left(V - V_{tot}\right)^2, \text{ iar } V_{tot}$$

este volumul total al obiectelor: $V_{tot} = \sum_{i=1}^{4} g_i \cdot vol_i$.

7. Exerciții

1. Explicați modul de funcționare al următorilor operatori: încrucișare (crossover), mutație. Căutați și înțelegeți diferite variante pentru operatorul de selecție. Explicați modul de funcționare al selecției de tip turneu și ruletă. Căutați și înțelegeți și alte variante de selecție.
2. Rulați programul prenzentat în capitolul 5. Observați de la ce generație nu se mai îmbunătățește soluția. Estimați câte evaluări s-au făcut până în acel moment. Rulați programul de mai multe ori și observați diferențele.
3. În cadrul programului prenzentat în capitolul 5, ajustați factorii care influențează modul de funcționare al algoritmilor genetici (mărimea populației, numărul de generații, parametrii funcției de performanță – scalarea diverselor contribuții de penalizare, etc.), și observați efectul acestor modificări asupra vitezei de convergență a algoritmului.
4. Implementați programul pentru problema prezentată în capitolul 6. Pentru implementare este necesar să modificați structura cromozomului și funcția de evaluare. Rulați programul, modificați parametrii de funcționare și testați diferențele.
5. Modificați implementarea pentru problema rucsacului: există *n* tipuri de obiecte, cu maxim 5 obiecte disponibile din fiecare tip. Problema este de a afla câte obiecte din fiecare tip trebuie puse în rucsac, cu aceleași obiective de a nu depăși volumul maxim și de a maximiza valoarea totală.

Laborator 2

Probleme matematice de optim local/global

1. Obiectivele laboratorului

- Însușirea modului de abordare a unei probleme matematice de optim local/global, folosind algoritmi genetici.
- Însușirea unor metode de evitare a blocării algoritmului într-o vecinătate a unui minim local
- Însușirea modului de abordare pentru găsirea celei mai bune aproximații polinomiale a unei funcții date
- Studiul convergenței, a preciziei soluției și a eficienței algoritmului

2. Minimul unei funcții matematice

2.1. Formularea problemei

De multe ori ne întâlnim în inginerie cu problema găsirii minimului (sau al maximului) unei funcții matematice, pentru care nu există întotdeauna o metodă analitică de abordare, mai ales când spatiul unde se caută soluția (mulțimea valorilor posibile) este prea mare. O problemă des întâlnită la algoritmii de optimizare este găsirea minimului global în cazul în care există și minime locale.

Abordăm mai jos o problemă matematică relativ simplă de optimizare, pentru a trece prin toți pașii rezolvării ei folosind algoritmi genetici:

Se dă funcția $f : R \rightarrow R$, $f(x) = x^4 - 2x^2 - x$, se cere să se găsească $x_m \in [-2, 2]$ pentru care f este minimă. Pentru x_m se cere o precizie de $\delta = 0.001$.

2.2. Rezolvarea problemei

Pentru rezolvarea problemei vom face referire la aspectele teoretice din laboratorul 1, capitolul 3. Algoritmul folosit va fi cel prezentat în Figura 1.

Stabilirea structurii cromozomului

Soluția acestei probleme (individul) este un număr real în intervalul [-2, 2]. Având în vedere precizia cerută δ, putem spune că spațiul de căutare al soluției este mulțimea

$S = \{-2, -1.999, -1.998, -1.997, ..., 1.999, 2\}$, cu $x_m \in S$

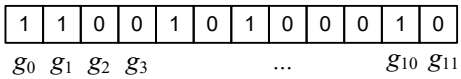
Cardinalul mulțimii S este $|S| = [2 - (-2)] / \delta + 1 = 4001$ valori.

Alegerea modului de construire a cromozomului s-ar putea face folosind o singură genă, având alela un număr fracționar aparținând mulțimii S, dar atunci nu ar exista suficientă diversitate, ceea ce ar împiedica operatorii de încrucișare și mutație să funcționeze eficient. Alegem deci genele de tip *boolean*, cu valorile în mulțimea $A = \{0, 1\}$. Putem privi deci fiecare genă ca un bit, iar întreg cromozomul va codifica un număr în intervalul cerut. Avem nevoie deci de:

$N = log_2 |S| = 11.96614$ biți

Nu putem folosi un număr de $N' = 11$ biți (gene de tip boolean), pentru că atunci precizia ar deveni $\delta' = [2 - (-2)] \ / \ 2^{N'} = 0.00195$, ceea ce nu este suficient. Se vor folosi deci $N = 12$ gene.

Genotipul este mulțimea G a tuturor cromozomilor posibili. Un exemplu de cromozom este următorul:



În figura de mai jos am indicat graficul funcției și noțiunile de bază (cromozom, mapping, individ, funcția de performanță, precizie, etc), cu referire directă pentru această problemă. Graficul funcției este arătat doar cu scop demonstrativ. Se observă că funcția prezintă un minim local și un minim global, diferite, incluse în intervalul [-2, 2].

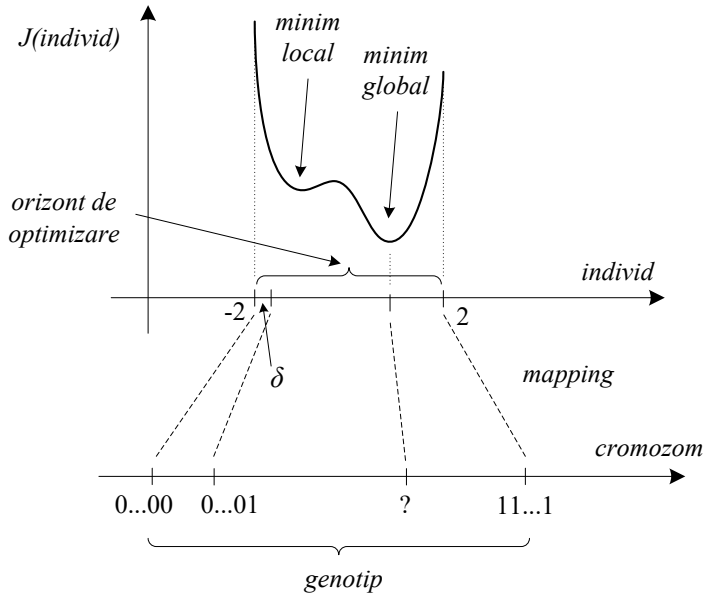


Figura 3. Minimul unei funcții matematice - concepte și noțiuni

Stabilirea funcției de mapping

Se caută o funcție de mapping $M : G \rightarrow [-2, 2]$. Cea mai simplă abordare este să considerăm mulțimea genelor cromozomului un număr binar (în baza 2), să-l transformăm în baza 10, și apoi să-l scalăm la intervalul cerut [5]. Avem deci:

$$M'(chr) = \frac{\sum_{k=0}^{11} g_k \cdot 2^k}{2^{12} - 1} \in [0, 1]$$

$$M(chr) = M'(chr) \cdot 4 - 2 = \frac{\sum_{k=0}^{11} g_k \cdot 2^k}{2^{12} - 1} \cdot 4 - 2, \text{ cu } M(chr) \in [-2, 2]$$

De exemplu, valoarea individului corespunzător cromozomului dat mai sus este:

$$M(chr_1) = \frac{3234}{2^{12}-1} \cdot 4 - 2 = 1.15897436 \in [-2, 2]$$

Stabilirea funcției de performanță

Funcția de performanță $J : [-2, 2] \rightarrow R_+$ poate fi considerată chiar funcția inițială, $J(x) = f(x) = x^4 - 2x^2 - x$, pentru că se caută chiar minimul acestei funcții. Totuși, constrângerile pachetului JGAP impun ca valoarea returnată de funcția de performanță să fie pozitivă, deci vom adăuga o valoare *bias* stabilită empiric la funcția J: $J(x) = f(x) + 5 = x^4 - 2x^2 - x + 5$

2.3. Implementare

Oferim mai jos codul aferent problemei de optimizare specificată.

Secvența de cod 1: Minimizarea unei funcții matematice

```
import org.jgap.*;
import org.jgap.impl.*;

public class FuncGrIV {
   private static final int MAX_ALLOWED_EVOLUTIONS = 10;
   public static void main(String[] args) throws
        InvalidConfigurationException{
     Configuration conf = new DefaultConfiguration();
     Configuration.resetProperty(Configuration.PROPERTY_FITEVAL_INST);
     //low value for fitness = better
     conf.setFitnessEvaluator(new DeltaFitnessEvaluator());
     conf.setPreservFittestIndividual(true);
     conf.setKeepPopulationSizeConstant(true);
     conf.setFitnessFunction(new FitnessFunctionGrIV());

     int nrGenes = 12;
     IChromosome sampleChromosome = new Chromosome(conf,
          new BooleanGene(conf), nrGenes);
     conf.setSampleChromosome(sampleChromosome);

     conf.setPopulationSize(20);
     Genotype population = Genotype.randomInitialGenotype(conf);

     for (int i = 0; i < MAX_ALLOWED_EVOLUTIONS; i++) {
       population.evolve();
       IChromosome bestChrSoFar = population.getFittestChromosome();
       System.out.println(i + ". " +
          "Fitness: " + bestChrSoFar.getFitnessValue() + ", " +
          "individual: " + FitnessFunctionGrIV.Mapping((bestChrSoFar)));
     }
   }
}

public class FitnessFunctionGrIV extends FitnessFunction {
   private final double POSITIVE_BIAS = 5;
   public double evaluate(IChromosome chr) {
     double individ = Mapping(chr);
     double fitness = Math.pow(individ,4) - 2 * Math.pow(individ,2) -
individ;
     return fitness + POSITIVE_BIAS;
   }
```

```
   public static double Mapping(IChromosome chr) {
      double base10 = 0;
      for (int i=0; i<chr.size(); i++) {
        boolean allele =
((Boolean)chr.getGene(i).getAllele()).booleanValue();
        if (allele)
          base10 += Math.pow(2, i);
      }
      double individ = base10 / (Math.pow(2, 12) - 1) * 4 - 2;
      return individ;
     }
}
```

2.4. Testarea programului și măsurarea performanțelor

Precizia soluției

Rulând programul, obținem destul de repede (de cele mai multe ori, la generația 6 sau 7) soluția $x_m \approx 1.1071$. Pentru verificarea preciziei, putem folosi resurse web (de exemplu www.wolframalpha.com) pentru găsirea soluției exacte x_m'. Pentru această problemă, $x_m' \approx 1.10716$, deci se confirmă faptul că soluția găsită este în intervalul de precizie cerut față de soluția exactă, adică $| x_m' - x_m| < \delta$

Convergența

Convergența este asigurată datorită configurației de păstrare a celui mai bun individ din fiecare generație și a operatorului de selecție. Pentru observarea directă a convergenței, recomandăm reprezentarea grafică a abaterii soluției în funcție de iterație.

În unele cazuri, datorită caracterului aleator al căutării, populația de indivizi rămâne „blocată" în vecinătatea minimului local. Aceasta este o problemă des întâlnită și dificil de rezolvat pe caz general la algoritmii de optimizare. Pentru a ocoli această situație se poate apela la unele artificii computaționale, dintre care amintim:

- asigurarea unei diversități suficiente în populația inițială astfel încât să existe cel puțin un individ în vecinătatea convexă a fiecărui minim local. Aceasta se poate asigura modificând mărimea populației.
- folosirea unui operator de selecție care menține în populație și indivizi mai puțin performanți, eventual folosind o probabilitate (de exemplu, selecția de tip turneu sau ruletă),
- generarea, în fiecare generație (sau când, timp de 5 sau 10 generații nu apare o îmbunătățire a soluției), a unui număr de soluții aleatoare care să aducă diversitate în populație. Această funcție este de obicei preluată de operatorul de mutație. Dacă nu sunt performante, soluțiile generate nu vor supraviețui în generația următoare.

Eficiența programului

Se observă ca s-a ales mărimea populației 20 și un număr de 10 de generații. Rezultă deci că, în total, se vor face un număr de $20 \times 10 = 200$ de evaluări într-un spațiu cu $2^{12} = 4096$ soluții posibile (deci aprox. 4.9 % dintre soluțiile posibile sunt evaluate). Putem observa deci că algoritmii genetici, deși au o componentă aleatoare specifică algoritmilor de *brute force*, sunt totuși mult mai performanți decât aceștia din urmă.

3. <u>Problemă propusă: aproximarea unei funcții cu o funcție polinomială</u>

Dezvoltarea funcțiilor în serie Taylor este deseori utilă pentru că reduce problema unei funcții complicate la un polinom ușor calculabil, util de exemplu când avem la dispoziție o putere de calcul redusă (microcontrolere, etc). Totuși, seria Taylor aproximează funcția *în jurul unui punct*, nu *pe un anumit interval*.

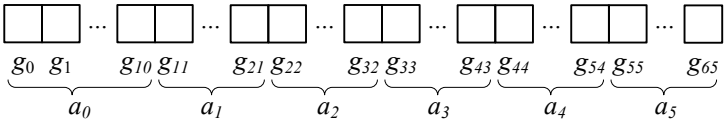
Ne propunem să găsim o funcție polinomială p: $R \rightarrow R$ de grad maxim 5, care aproximează cel mai bine funcția f: $R \rightarrow$ [-1, 1], f(x) = $sin(x)$, pe intervalul [0, π].

<u>Stabilirea structurii cromozomului</u>

Notăm funcția polinomială de grad 5 cu p(x) = $a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$. Problema devine deci de a găsi coeficienții a_5, a_4, a_3, a_2, a_1, a_0 pentru care aproximarea este optimă (individul este deci funcția $p(x)$). Similar cu problema precedentă, considerăm o precizie $\delta = 0.01$ pentru coeficienți (două zecimale), și căutăm coeficienții numai în intervalul [-10, 10]. Pentru fiecare coeficient avem nevoie deci de:

$$N = \log_2 \frac{10-(-10)}{\delta} = \log_2 2000 \le 11 \text{ biți.}$$

Cromozomul va avea în consecință 66 de gene cu alela în mulțimea {0, 1} (gene de tip *boolean*), grupate câte 11 pentru fiecare coeficient în parte, astfel:



<u>Stabilirea funcției de mapping</u>

Căutăm o funcție de mapping M : $G \rightarrow P$, unde G este genotipul care a fost descris mai sus și P este mulțimea tuturor polinoamelor de grad 5. Considerăm cromozomul format din 6 secvențe de câte 11 biți (corespunzătoare fiecărui coeficient), iar apoi, în mod similar cu problema precedentă, transformăm fiecare secvență în baza 10 și o scalăm la intervalul [-10, 10]. Avem deci:

$$M(chr) = \left(\frac{\sum_{k=0}^{10} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^5 + \left(\frac{\sum_{k=11}^{21} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^4 + \ldots + \left(\frac{\sum_{k=55}^{65} g_k \cdot 2^k}{2^{11}-1} \cdot 20 - 10 \right) \cdot x^0$$

<u>Stabilirea funcției de performanță</u>

Performanța unui polinom $p(x)$ se măsoară în funcție de ”cât de bine aproximează funcția dată”. Mai exact, abaterea dintre $p(x)$ și $sin(x)$ trebuie să fie minimă în orice punct pe intervalul [0, π]. Aceasta o putem formula matematic astfel: căutăm coeficienții $a_5, \ldots, a_0$ pentru care se atinge $M_a = \min_{a5,\ldots,a0} \int_0^{\pi} \left| p(x) - \sin(x) \right| dx$

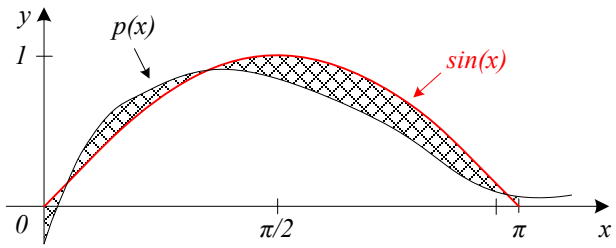


Figura 4. Funcția obiectiv și funcția polinomială

Integrala de mai sus reprezintă de fapt penalizarea pe care o aplicăm unui polinom $p(x)$. Diferența este luată în modul pentru că trebuie penalizat și cazul în care $p(x)$ este sub graficul $sin(x)$ (adică o distanță negativă). Problema de optimizare este deci una de minimizare a funcției de performanță. Pentru a facilita calculul performanței unui polinom, discretizăm integrala și o aproximăm cu o sumă de penalizări individuale. Considerăm un număr de 1000 de puncte intervalul [0, π] și calculăm diferența între valoarea polinomului și funcția dată numai în acele puncte.

$$J(p(x)) = \sum_{k=0}^{1000} \left| p(\frac{k \cdot \pi}{1000}) - \sin(\frac{k \cdot \pi}{1000}) \right|$$

Evident, în cazul acestei probleme este nevoie de o populație mai numeroasă și un număr de generații ridicat din cauză că spațiul de căutare este mai mare.

4. Exerciții

1. Rulați programul dat în capitolul 2. Observați precizia soluției și convergența algoritmului. Modificați parametrii programului și notați observațiile.
2. Construiți un program care să reprezinte grafic abaterea soluției găsite în fiecare generație la exercițiul anterior în funcție de numărul generației. Analizați și discutați graficul. Măsurați timpul mediu necesar rulării pentru o generație.
3. Reduceți mărimea populației și generați toată populația inițială în vecinătatea convexă a minimului local. După câte generații depășește algoritmul zona de minim local și din ce cauză ? Cât mai durează până găsește minimul global ?
4. Implementați un program Java care să rezolve problema de la capitolul 3 folosind algoritmi genetici. Pe baza modelului dat în capitolul 2, modificați structura cromozomului și funcția de performanță. Pentru aceasta din urmă, implementați mai întâi o metodă care calculează valoarea unui polinom dat pentru un argument x_0.
5. Analizați spațiul de căutare al soluției pentru problema din capitolul 3. Care este eficiența algoritmului ? Încercați mai multe variante pentru numărul de generații și mărimea populației. Cum afectează fiecare dintre ele performanța algoritmului ?

Laborator 3

Probleme de optim în automatică

1. Obiectivele laboratorului

- Însușirea unei metode de calcul a mărimii de control pentru obținerea unei traiectorii date pentru ieșirea unui sistem, folosind algoritmi genetici
- Însușirea unei metode de identificare a sistemelor, folosind algoritmi genetici
- Exerciții

2. Studiu de caz I: controlul unui sistem pentru a obține traiectoria dată

În teoria controlului se abordează de multe ori problema calculului mărimii de control a unui sistem astfel încât ieșirea să fie cât mai aproape de o referință dată. O problemă mai complexă și dificil de rezolvat pe căi clasice este aceea de a calcula mărimea de control care produce o traiectorie a ieșirii cât mai aproape de o traiectorie referință [6],[7]. Lungimea intervalului în care traiectoria referință trebuie urmărită în mod optim se numește *orizont de optimizare*.

2.1. Formularea problemei

Se dă un sistemul de gradul II în spațiul stărilor, cu o intrare, două stări și o ieșire. Valorile inițiale ale stărilor sunt zero, iar mărimea de intrare $u(k)$ este limitată în intervalul [-2, 2].

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 0.7 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 5 \\ 3 \end{bmatrix} \cdot u(k) \\ y(k) = \begin{bmatrix} 0.3 & -0.7 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \end{cases}$$

Se consideră orizontul de optimizare [0, 13]. Se cere să se calculeze mărimea de intrare $u(k)$ pentru care mărimea de ieșire se apropie cât mai mult de referința indicată cu roșu în figura de mai jos.

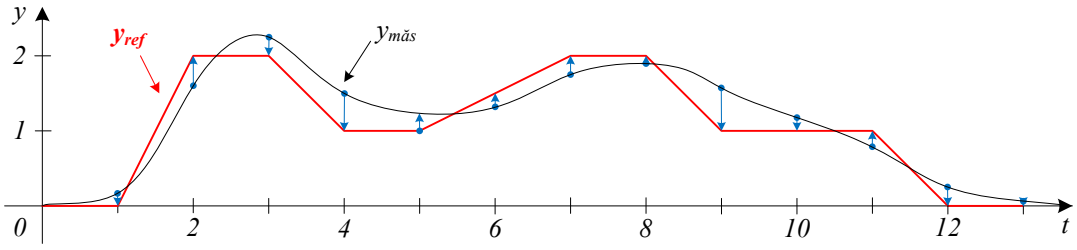
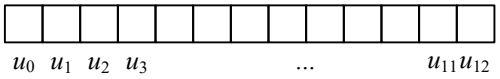


Figura 5. Controlul unui sistem pentru a obține traiectoria dată

2.2. Rezolvarea problemei

Soluția problemei (individul) este un vector $U = [u_0, u_1, ..., u_{12}]$ de valori reale, care conține fiecare mărime de intrare pentru intervalul [0, 12]. Întradevar, orizontul de optimizare este [0, 13], dar se observă că valoarea u_{13} nu își produce efectele în intervalul de optimizare, deci este irelevantă din punctul de vedere al problemei.

Pentru această problemă, vom considera un cromozom compus din 13 gene, așa cum este indicat mai jos. Valoarea pentru fiecare alelă este un număr real (`DoubleGene`) în intervalul dat, deci $u_k \in G = [-2, 2]$. Funcția de mapping va fi deci funcția identică.



Funcția de performanță vizează, ca și în capitolul 3 din laboratorul precedent, diferența între valoarea *y* măsurată și referința dată. Pentru că sistemul este discret, se vor considera numai valorile care corespund momentelor de eșantionare $t = 0, 1, 2, \ldots, 13$, deci segmentele indicate cu albastru în figura anterioară.

2.3. Implementare

Oferim mai jos codul relevant pentru funcția de performanță. Programul principal rămâne similar cel prezentat în laboratorul anterior.

În secvența de cod de mai jos, metoda `Mapping(chr)` extrage din obiectul de tip `IChromosome` valorile alelelor și returnează un array de 13 valori double. Metoda `GetY(u)` este practic simulatorul sistemului: pe baza a 13 valori de intrare, *u*, calculează iterativ mai întâi stările x_1, x_2 și apoi valorile de ieșire *y*. Aceste două metode trebuie implementate.

Secvența de cod 1. Funcția de performanță

```
import org.jgap.*;
public class FitnessFunctionTrajectory extends FitnessFunction {
  private static final double[] yref = { 0, 0, 2, 2, 1, 1, 1.5, 2, 2,
1, 1, 1, 0, 0 };
  private static final int optimizationHorizon = 13;
  public double evaluate(IChromosome chr) {
    double[] u = Mapping(chr);
    double[] y = GetY(u);
    double errorSum = 0;
    for (int i=0; i<optimizationHorizon; i++)
      errorSum += Math.abs(y[i]-yref[i]);
    return errorSum;
  }

  private double[] Mapping(Ichromosome chr) {
    //add Mapping(chr) method
  }

  private double[] GetY(double[] u) {
    //add GetY(u) method
  }
}
```

2.4. Testare și concluzii

Observații privind structura cromozomului și operatorii genetici

La problemele din laboratorul precedent am considerat cromozomul ca având gene de tip boolean; practic, am distribuit fiecare necunoscută care trebuie aflată pe parcursul a mai multe gene (vezi laborator 2, capitolul 3). Numărul de gene devine astfel mai mare (deci implică mai multe calcule), dar operatorul de încrucișare este mai eficient [8]. Nu există o regulă care dictează în ce cazuri se abordează o strategie sau alta, decizia ține mai mult de experiența programatorului.

În general, operatorul de încrucișare este cel care transformă cea mai mare parte a cromozomilor, ajutând genele bune să se perpetueze de la o generație la alta. Operatorul de mutație este responsabil pentru introducerea, din când în când, a unor noi caracteristici în bazinul de cromozomi, dar și pentru scoaterea sistemului intr-o eventuală stare de optim local, dar nu global.

În cazul folosirii genelor de tip număr real, nu se poate ajusta precizia folosită, ea fiind implicit precizia (variabilă!) a numerelor reale în limbajul de programare ales.

Parametrii de testare

În cazul acestei probleme, vom stabili mărimea populației mai mare decât în alte cazuri pentru a introduce un grad mare de hazard (engl. *randomness*) de la început, pentru a compensa ineficiența relativă a operatorului de încrucișare. Vom stabili populația ca având 1000 de indivizi și vom lăsa algoritmul să ruleze 200 de generații.

Limitările abordării

Pentru a garanta convergența și comportamentul consistent al algoritmului, este necesar ca, ori de câte ori se evaluează același cromozom (în aceeași generație sau în generații diferite) să obținem aceeași valoare a performanței. Dacă sistemul are o componentă stohastică sau există perturbații aleatoare care nu pot fi controlate (chiar dacă ele pot fi măsurate), este evident că evaluarea nu poate fi făcută în mod consistent. În acest caz, putem recurge de exemplu, la simularea repetată a sistemului (cu diferite perturbații) și returnarea unui indice de performanță mediu.

O altă limitare o constituie sistemele continue, pentru care mărimea de control nu poate fi mapată pe un cromozom cu componente discrete. În acest caz, putem recurge la aproximarea sistemului prin discretizare.

3. Studiu de caz II: identificarea sistemelor folosind algoritmi genetici

Metodele performante de identificare a sistemelor sunt extrem de laborioase și fac uz de multe detalii care sunt de obicei greu de reținut, mai ales dacă nu se lucrează cu ele sistematic [9]. Prezentăm în continuare o variantă de identificare a sistemelor cu algoritmi genetici, care primează prin ușurința abordării. Ca în orice problemă, parametrii necunoscuți trebuie codificați în cromozom și evaluarea se face prin simularea sistemului.

3.1. Formularea problemei

Se dă un sistem de gradul II ca o cutie neagră (*blackbox*), deci nu se cunoaște decât comportamentul exterior (mărimea măsurată y, în funcție de intrarea u). Se cere să se calculeze parametrii unui sistem de gradul II în spațiul stărilor care aproximează cât

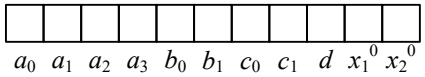
mai bine sistemul dat. Există o limitare pe mărimea de intrare $u \in [0, 5]$, iar orizontul de optimizare este $[0, \tau]$

3.2. Rezolvarea problemei

Considerăm modelul unui sistem de gradul II în spațiul stărilor în formă generală. Necunoscutele sunt coeficienții $a_0, \ldots, a_3, b_0, b_1, c_0, c_1, d$, numere reale:

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} a_0 & a_1 \\ a_2 & a_3 \end{bmatrix} \cdot \begin{bmatrix} x_2(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \cdot u(k) \\ y(k) = \begin{bmatrix} c_0 & c_1 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + d \cdot u(k) \end{cases}$$

Ca și în exemplul anterior, vom considera cromozomul ca fiind format din gene cu alele de tip număr real (`DoubleGene`), și vom limita intervalul genotip G = [-10, 10]. Este nevoie deci de 9 gene pentru parametri și încă 2 gene pentru valorile stărilor inițiale, x_1^0, x_2^0, cu același genotip G. Individul va fi modelul sistemului în spațiul stărilor.



Aranjamentul pentru evaluarea sistemului este arătat în figura de mai jos. Pentru a calcula performanța unui model găsit vom lua suma integrată a erorii între $y_{măsurat}$ și $y_{estimat}$ pe întreg orizontul de optimizare τ. Vom considera în schimb pătratul erorii, pentru a penaliza progresiv diferențiat cazurile în care comportamentul modelului se abate puternic de la cel al procesului pe care îl identificăm. Avem deci, în varianta discretă: $J = \sum_{k=0}^{\tau} \left(y_m(k) - y_{est}(k)\right)^2$. Valorile ieșirii $y(k)$, cu $k \in [0, \tau]$ vor fi calculate ca și în exemplul precedent, folosind o metodă căreia îi transmitem ca parametrii modelul și valorile mărimilor de intrare u, `double[] GetY(a, b, c, d, u)`.

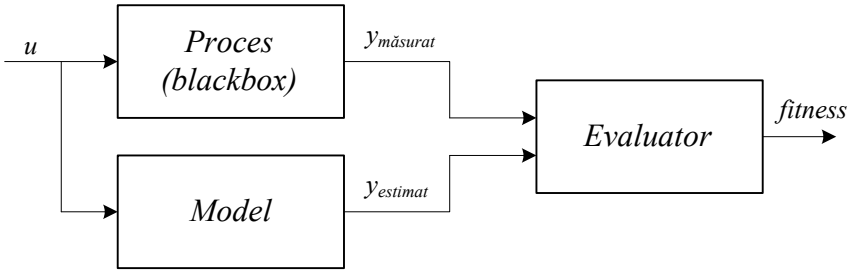


Figura 6. Aranjament pentru identificarea sistemului

Problema de optimizare este aceea de a *minimiza* funcția de performanță J pentru *toate valorile posibile* ale intrării u. Dacă folosim o singură valoare a intrării, de exemplu semnalul treaptă, vom obține un model al sistemului care funcționează bine numai pentru acea valoare a intrării u. Pentru simplitate, vom considera totuși doar 2 cazuri: treaptă, și rampă. Vom avea deci:

$$J = \sum_{k=0}^{\tau} \left(y_m^{step}(k) - y_{est}^{step}(k) \right)^2 + \sum_{k=0}^{\tau} \left(y_m^{ramp}(k) - y_{est}^{ramp}(k) \right)^2$$

Vom alege τ = 20s, iar pentru semnalul treaptă: $u(k) = 0$ pentru $k \leq 5$ și $u(k) = 1$ pentru $k > 5$. Sistemul trebuie să păstreze ieșirea și în lipsa unui semnal de intrare.

Pentru semnalul rampă: $u(k) = 0.25 * k$, factorul de scalare având rolul de a păstra valoarea intrării în intervalul dat.

Nu este întotdeauna evident ce influență au semnalele de intrare stabilite asupra performanței modelului obținut (sau a soluției problemei, pe caz general). Ca o regulă generală, o soluție generată va fi performantă doar pentru cazurile în care ea a fost evaluată (testată) în funcția de performanță. Ține de experiența programatorului găsirea acelor scenarii de testare a soluției potențiale care să acopere toate cazurile necesare în problemă.

4. Exerciții

1. Pentru exemplul din capitolul 2, implementați un program Java folosind pachetul JGAP, care calculează mărimea de intrare $u(k)$, k=0,1,...,12 cu algoritmi genetici. Pentru aceasta, completați metodele `Mapping(chr)` și `GetY(u)`, oferite în secvența de cod dată.
2. Cum ar trebui modificată funcția de evaluare a programului anterior dacă sistemul primește o perturbație aleatoare în intervalul [-0.2, 0.2] care acționează direct pe starea x_1 ? Realizați modificarea și evaluați convergența și consistența algoritmului.
3. Înlocuiți numărul fix de generații cu următoarea condiție de oprire: dacă pe parcursul a 10 generații nu se înregistrează o creștere a performanței mai mare decat 5%, algoritmul se oprește. Câte generații rulează algoritmul, în medie ?
4. Pentru exemplul din capitolul 3, implementați un program Java care să rezolve identificarea sistemului folosind algoritmi genetici. Pentru sistemul necunoscut (*blackbox*) se va folosi sistemul din capitolul 2, cu valorile inițiale ale stărilor: $x_1^0 = 2$, $x_2^0 = -2$.
5. Testați sistemul identificat cu un semnal periodic, apoi unul aleator. Cum răspunde sistemul ? Cum se poate modifica funcția de evaluare pentru a cuprinde și acest caz ?

Laborator 4

Determinarea parametrilor pentru controlul cu logica fuzzy

1. Obiectivele laboratorului

- Recapitularea conceptelor: control fuzzy, Fuzzy Logic Enhanced Time Petri Nets (FLETPN)
- Însușirea unei metode de generare a regulilor pentru un sistem de control cu logica fuzzy, cu ajutorul algoritmilor genetici
- Însușirea unei metode de generare a tabelelor de reguli fuzzy logic pentru un sistem FLETPN cu specificații date
- Studiul efectului pe care îl are modul de evaluare asupra regulilor fuzzy generate.

2. Recapitularea conceptelor legate de logica fuzzy

2.1. Controlul fuzzy

Ființele umane folosesc o logică intuitivă bazată pe valori discrete (*fuzzy*) și reguli fuzzy de forma dată mai jos, unde x_1, x_2, y_1, y_2 sunt valori fuzzy care aparțin mulțimilor finite X_1, X_2, Y_1, Y_2.

IF (x_1 IS X_1 AND x_2 IS X_2 …) THEN (y_1 IS Y_1 AND y_2 IS Y_2 …)

Pentru a putea folosi reguli fuzzy la controlul unui sistem automat vom folosi structura de mai jos [10], unde sunt reprezentate operațiile necesare: *fuzzyficare* (transformare din valori numerice în valori fuzzy), *aplicarea regulilor* de control *FLR* pe valori fuzzy, *defuzzyficare* (transformare din valori fuzzy în valori numerice).

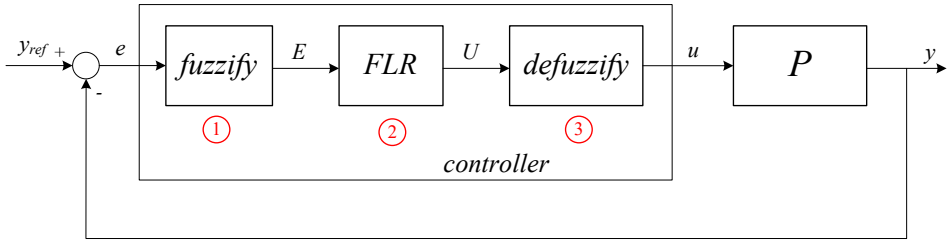


Figura 7. Controlul fuzzy al unui proces

Pentru *fuzzyficare* și *defuzzyficare* vom folosi *funcțiile de apartenență* date mai jos (*NL* – negative large, *NM* – negative medium, *ZR* – zero, *PM* – positive medium, *PL* – positive large), cu valorile numerice corespunzătoare, în ordine: -1, -0.5, 0, 0.5, 1. Valorile fuzzy vor fi reprezentate ca vectori ce conțin 5 grade de apartenență, de exemplu $x_1' = [0, 0, \mu_{ZR}', \mu_{PM}', 0]$.

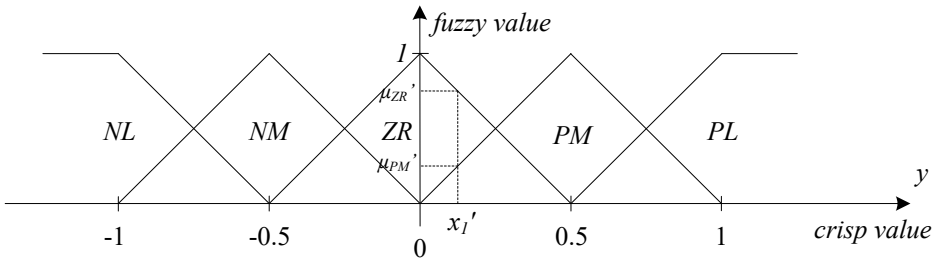


Figura 8. Funcții de apartenență (membership functions)

În figura de mai jos sunt reprezentate regulile fuzzy pentru un sistem cu 2 intrări. Pentru x_1 între *ZR* si *PM*, iar x_2 între *NM* și *ZR*, avem 4 reguli active, indicate în tabel. Pe baza gradului de îndeplinire a fiecărei reguli, se calculează mărimile de ieșire. Pentru un sistem cu o singură intrare, tabelul de reguli devine un simplu vector.

x_2 \ x_1	*NL*	*NM*	*ZR*	*PM*	*PL*
NL	(*PL*, *PL*)	(*PL*, *PM*)	(*PL*, *ZR*)	(*PL*, *NM*)	(*PL*, *NL*)
NM	(*PM*, *PL*)	(*PM*, *PM*)	(*PM*, *ZR*)	(*PL*, *NM*)	(*PM*, *NL*)
ZR	(*ZR*, *PL*)	(*ZR*, *PM*)	(*ZR*, *NL*)	(*ZR*, *NM*)	(*ZR*, *NL*)
PM	(*NM*, *PL*)	(*NM*, *PM*)	(*NM*, *ZR*)	(*NM*, *NM*)	(*NM*, *NL*)
PL	(*NL*, *PL*)	(*NL*, *PM*)	(*NL*, *ZR*)	(*NL*, *NM*)	(*NL*, *NL*)

Figura 9. Reguli fuzzy reprezentate ca tabel (două intrări: x_1, x_2)

2.2. Fuzzy Logic Enhanced Time Petri Nets (FLETPN)

Structurile FLETPN [11] sunt rețele Petri modificate în care jetoanele au în componență o valoare fuzzy cu 5 grade de apartenență, ca mai sus. Tranziția de bază a unei rețele FLETPN implementează un set de reguli fuzzy (FLRS). Tranziția devine executabilă atunci când cel puțin o regulă din tabelul aferent este activă. Factorii de amplificare w_1, $w_2 \in$ [-10, 10] scalează mărimile de intrare.

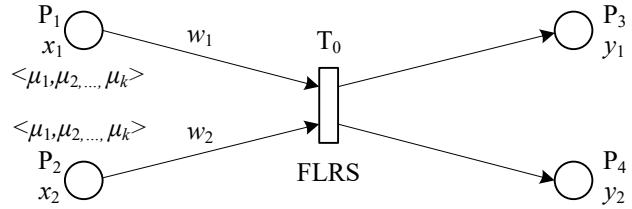


Figura 10. Exemplu de tranziție FLETPN

În plus față de logica fuzzy standard, modelul FLETPN introduce o nouă valoare fuzzy, „φ” (*FF*), care semnifică lipsa jetonului, fie la intrare, fie la ieșire. Cu ajutorul acestei valori, se poate descrie de exemplu un arc inhibitor. Valoarea fuzzy FF este tratată identic ca și celelalte valori în tabelul de reguli.

3. Studiu de caz I: stabilirea regulilor fuzzy logic de control pentru un sistem simplu

Determinarea regulilor de control fuzzy care oferă performanțe ridicate este o problemă în sine. Sunt necesare cunoștințe ample despre sistemul care trebuie controlat și implică de obicei experiența experților care descriu tabelele de reguli. O abordare mai simplă și mai directă o constituie algoritmii genetici, care implică cunoștințe minime despre sistem, cromozomul codificând direct necunoscuta (regulile de control) [12], [13].

3.1. Formularea problemei

Se dă modelul matematic al unui sistem cu intrări (u_i) și ieșiri (y_i). Se cere stabilirea regulilor de control fuzzy pentru care sistemul respectă o referință dată.

De exemplu, vom folosi modelul unui sistem de gradul II în spațiul stărilor care modelează un cuptor cu gaz. Stările sistemului reprezintă concentrația de oxigen din camera de ardere (x_1), respectiv temperatura (x_2), iar intrările sunt debitul de gaz (u_1), respectiv aportul de oxigen (u_2). Ieșirea reprezintă temperatura în cuptor.

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.2 & -0.4 \\ -0.1 & 0.9 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ -0.1 & -0.1 \end{bmatrix} \cdot \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \\ y(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \end{cases}$$

Vom calcula regulile de control pentru un sistem de control fuzzy care comandă sistemul pentru a respecta o referință dată. Starea inițială este $x_1(0) = 1$, $x_2(0) = 0$ și se consideră intrările, stările și ieșirea sistemului limitate în intervalul [-1, 1] (în caz contrar, ele pot fi scalate folosind niște factori constanți).

3.2. Rezolvarea problemei

În cele ce urmează vom considera funcțiile de apartenență fuzzy din Figura 10, deci vom folosi 5 nivele fuzzy: *NL*, *NM*, *ZR*, *PM*, *PL*. Vom folosi ca și arhitectură a sistemului cea din figura de mai jos, observând că mărimea de control are două componente, u_1 și u_2. Convenția de notație este că valorile numerice sunt reprezentate cu litere mici, iar cele fuzzy cu majuscule.

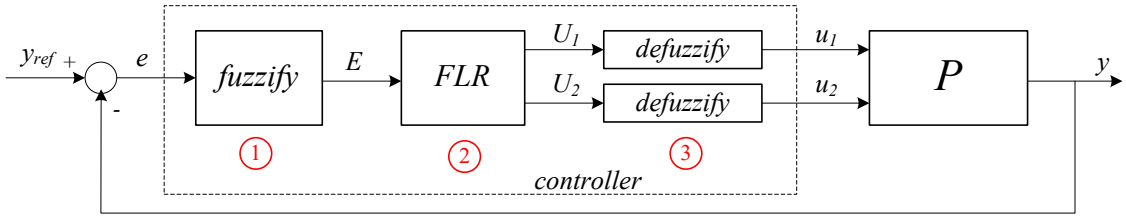


Figura 11. Arhitectura sistemului cu control fuzzy

Tabelul de reguli implementate de blocul *FLR* este de fapt un vector de 5 perechi, pentru că avem o singură intrare, *E*, și două ieșiri, U_1 și U_2. Un exemplu valid de reguli este indicat mai jos. Necunoscutele problemei sunt regulile de control înscrise în tabel, deci acestea vor fi codificate în cromozom [12]. Tabelul de reguli este forma soluției căutate (individul).

E	NL	NM	ZR	PM	PL
(U_1,U_2)	(PL, PL)	(ZR, PM)	(NL, ZR)	(PL, NM)	(PL, PL)

Pentru a stabili structura cromozomului vom partitiona tabelul de perechi în două tabele distincte (pentru U_1, respectiv U_2), rezultând deci un vector de 10 gene (5 pentru U_1 și 5 pentru U_2). Genele conțin numere întregi în intervalul [1, …, 5], care codifică cele 5 nivele fuzzy: *NL*, *NM*, *ZR*, *PM*, *PL*. O alternativă este se implementeze o nouă clasă specială pentru acest tip de gene (`FuzzyGene`).

Funcția de performanță practic simulează sistemul și penalizează eroarea totală a ieșirii y față de referință y_{ref}. Cerința problemei este ca sistemul să respecte orice referință dată, deci funcția de performanță ar trebui să testeze toate valorile posibile pentru referință în intervalul [-1, 1].

3.3. Implementare

Oferim mai jos secvența de cod care implementează funcția de performanță.

Secvența de cod 1: Funcția de performanță pentru sistemul fuzzy

```
public class FuzzFitnessFunction extends FitnessFunction {
  private static final double[] yrefValues = { -1, -0.5, 0, 0.5, 1};
  private static final int optimizationHorizon = 20;

  public double evaluate(IChromosome chr) {
    OneXTwoTable FLRS = Mapping(chr);
    double errorSum = 0;
    //should use every possible setpoint
    for (double yref : yrefValues){
      double[] y = GetY(FLRS, yref);
      for (int i=0; i < optimizationHorizon; i++)
        errorSum += Math.abs(y[i]-yref);
    }
    return errorSum;
  }
}
```

În secvența de cod dată se folosește pachetul JGAP, precum și utilitarul FuzzyP, disponibil la adresa: https://github.com/AttilaOrs/FuzzP/tree/master/fatJar, care implementează logica fuzzy. Fișierul *.jar* trebuie adăugat ca referință în proiect. Diagrama claselor pentru cele mai importante componente ale acestei biblioteci este prezentată mai jos. Clasa indicată cu roșu (`FuzzFitnessFunction`) este cea dată în secvența de cod de mai sus și trebuie implementată astfel: metoda `Mapping(chr)` preia din cromozom regulile de control și construiește un obiect `OneXTwoTable`, iar metoda `GetY(FLRS, yref)` calculează ieșirea y prin simularea sistemului (figura 11) pe întreg orizontul de optimizare.

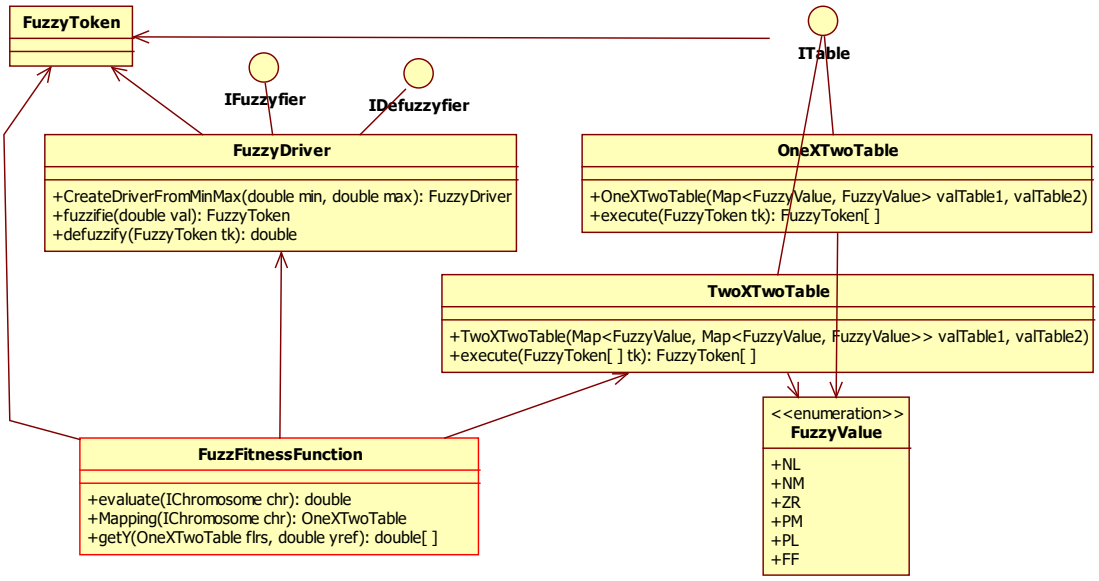


Figura 12. Diagrama claselor pentru pachetul FuzzyP

3.4. Testare și concluzii

Precizia soluției

Algoritmul genetic poate stabili valoarea corectă numai pentru acele reguli care au fost activate la un moment dat în cadrul evaluării performanței. Regulile care nu au fost activate niciodată nu au nici un efect asupra performanței, deci valoarea lor va fi aleatoare. Din acest motiv s-a recurs la simularea repetată a sistemului pentru mai multe valori ale referinței. Pentru simplitate, în funcția de performanță au fost testate, ca și referință, doar valorile care corespund lui *NL*, *NM*, *ZR*, *PM* și *PL*. Aceste valori acoperă intervalul specificat [-1, 1].

Convergența și eficiența algoritmului

În acest caz, spațiul de căutare are mărimea $|S| = 5^{10}$ (5 posibilități pentru fiecare dintre cele 10 gene). Mărimea populației trebuie să fie suficient de mare pentru a oferi varietate bazinului genetic, iar numărul de generații trebuie să fie suficient de mare încât fiecare genă (statistic vorbind) să poată fi atinsă de mutație sau încrucișare. Totuși, trebuie ținut cont că pentru fiecare evaluare de cromozom se fac practic 5 simulări ale sistemului, pe orizontul de optimizare. Recomandăm stabilirea mărimii populației și a numărului de generații astfel încât să se evalueze în total aproximativ 2-3% din spațiul de căutare.

Considerații despre timpul de execuție

Folosind această abordare, nici nu este necesară cunoașterea modelului matematic al sistemului. De exemplu, în cazul unui sistem software de tip *blackbox*, se poate recurge în loc de simulare la activarea efectivă a sistemului (prin apelarea metodelor unei aplicații externe). În mod similar se poate proceda în cazul unor sisteme fizice rapide. În ambele cazuri, întârzierile de comunicație și ale sistemului pot fi relevante și trebuie luate în considerare.

4. Studiu de caz II: stabilirea regulilor pentru un sistem FLETPN

În cazul unor sisteme mai complexe, de exemplu atunci când sunt necesare mai multe tabele de reguli, aceste tabele pot fi liniarizate și concatenate pentru a forma cromozomul. Prezentăm mai jos un exemplu de componentă FLETPN pentru care trebuie stabilite regulile de inferență fuzzy din tranziții.

4.1. Formularea problemei

Se dă componenta FLETPN din figura de mai jos, cu 3 porturi de intrare (P_1, P_2, P_3) și 2 porturi de ieșire (T_2, T_3). Valorile jetoanelor corespunzătoare porturilor sunt x_1, x_2, δ, pentru intrări și y_1, y_2 pentru porturile de ieșire. Toate jetoanele sunt de tip fuzzy cu mulțimea valorilor $S = \{NL, NM, ZR, PM, PL, FF\}$. Componenta reprezintă un comparator cu histerezis, și funcționează astfel:

- dacă $x_1 - x_2 > \delta$, atunci $y_1 = 1$ și $y_2 = \varphi$ (deci doar T_2 este activată)
- dacă $x_2 - x_1 > \delta$, atunci $y_1 = \varphi$ și $y_2 = -1$ (deci doar T_3 este activată)
- dacă $|x_1 - x_2| \leq \delta$, atunci $y_1 = 1$ și $y_2 = -1$ (ambele tranziții de ieșire sunt activate)
- dacă lipsește una din mărimile x_1, x_2 (au valoare *FF*), atunci $y_1 = \varphi$ și $y_2 = \varphi$ (nici o tranziție de ieșire nu este activată), iar dacă lipsește δ, acesta va fi considerat $\delta = 0$

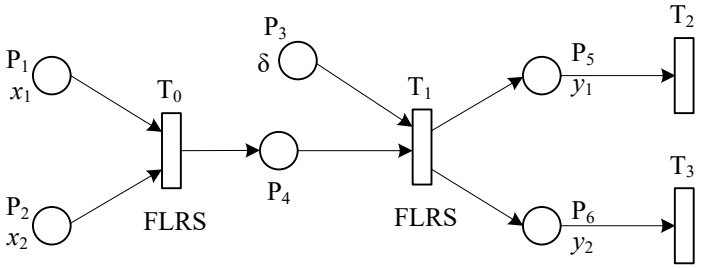


Figura 13. Modelul componentei ca rețea de tip FLETPN

Se cere să se găsească tabelele de reguli FLRS pentru tranzițiile interne ale componentei (T_0 și T_1) astfel încât să se respecte specificațiile comportamentale date.

4.2. Rezolvarea problemei

Pentru mulțimea valorilor fuzzy S dată, tabelul de reguli FLRS al tranziției T_0 conține $6 \times 6 = 36$ de reguli (pentru o singură ieșire), iar pentru tranziția T_1 care are două ieșiri, $2 \times 36 = 72$ de reguli, deci în total 108 reguli. În consecință vom considera un cromozom cu 108 gene de tip număr întreg (`IntegerGene`, cu intervalul [1, …, 5], similar cu exemplul anterior).

Funcția de performanță penalizează, ca de obicei, diferența dintre comportamentul real al sistemului și cel referință. Ca și în exemplul anterior, ar fi necesar să testăm în funcția de performanță toate combinațiile posibile ale valorilor de intrare x_1, x_2, δ, pentru a activa fiecare regulă cel puțin o dată. În cazul nostru, vom parcurge doar cazurile aferente valorilor fuzzy și vom penaliza diferența dintre valorile y_1, y_2 și referințele lor din formularea problemei.

4.3. Implementare

Oferim mai jos funcția de performanță recomandată.

Secvența de cod 2: Funcția de performanță pentru componenta FLETPN

```
public class FLETPNFitnessFunction extends FitnessFunction {
  private static final double[] fuzValues = { -1, -0.5, 0, 0.5, 1};

  public double evaluate(IChromosome chr) {
     TwoXOneTable FLRS1 = Mapping1(chr);
     TwoXTwoTable FLRS2 = Mapping2(chr);

     double errorSum = 0;
     //should use every possible combination of inputs
     for (double x1 : yrefValues)
        for (double x2 : yrefValues)
           for (double d : yrefValues){
              FuzzyToken[] yref = GetYRef(x1, x2, d);
              FuzzyToken[] y = GetY(FLRS1, FLRS2, x1, x2, d);
              errorSum += GetDiff(yref[0],y[0]) + GetDiff(yref[1],y[1]);
           }
     return errorSum;
  }
}
```

Metoda `GetYRef` returnează valorile fuzzy ale lui y_1 și y_2 ce vor fi folosite ca referință, conform specificațiilor problemei. Metoda `GetY` calculează valorile fuzzy ale lui y_1 și y_2 aplicând intrările `x1`, `x2` și `d` fuzzyficate, pe tabelele `FLRS1` și apoi `FLRS2`.

Metoda `GetDiff` compară valorile jetoanelor date ca argumente și returnează diferența absolută dintre ele (le defuzzyfică, apoi calculează diferența în modul).

5. Exerciții

1. Implementați un program Java pentru problema prezentată în capitolul 3. Stabiliți mărimea populației și numărul de generații conform indicațiilor. Testați programul și observați precizia soluției.
2. Eliminați unele dintre valorile referinței (`yrefValues`). Care reguli nu vor fi activate acum ? Rulați programul de mai multe ori și verificați faptul că regulile neactivate niciodată sunt generate aleator.
3. Implementați programul pentru studiul de caz II, care stabilește regulile de inferență fuzzy aferente tranzițiilor pentru componenta dată. Testați programul și evaluați performanța stabilirii regulilor.
4. Notați tabloul optim de reguli găsit la exercițiul 1. Până acum, am folosit pentru simplitate aceleași funcții de apartenență pentru toate mărimile, dar aceasta nu este întotdeauna varianta reală. Implementați acum un program Java care va găsi cele mai bune funcții de apartenență (în locul celor din Figura 8). În esență, trebuie găsite valori numerice pentru nivelele fuzzy *NL*, *NM*, *ZR*, *PM*, *PL* pentru fiecare dintre mărimile de intrare și de ieșire: $e = y_{ref} - y$, u_1, u_2 (deci în total 15 valori).
   Atenție ! Valorile pentru nivelele fuzzy trebuie să fie în ordine, deci nu este permis de exemplu ZR > PM. Pentru construirea funcțiilor de apartenență se va folosi clasa `TriangleFuzzifier`.

Laborator 5

Alocarea resurselor unui sistem (controlul sistemului de trafic feroviar)

1. Obiectivele laboratorului

- Însușirea unei metode de determinare a alocării resurselor unui sistem, folosind algoritmi genetici. Studiu de caz: generarea orarului trenurilor (*train schedule*),
- Însușirea unei metode de generare a rutelor trenurilor în sistemul feroviar, folosind algoritmi genetici. Generalizare pentru rute într-un graf oarecare.
- Însușirea modului de lucru pentru probleme cu constrângeri.

2. Studiu de caz I: determinarea alocării resurselor unui sistem folosind algoritmi genetici (generarea orarului trenurilor)

Alocarea de resurse într-un sistem este o problemă cu constrângeri multiple, referitoare la interdependențele care există între resurse și componentele care le utilizează, precum și între acestea și aspectul temporal. Deși există algoritmi de alocare a resurselor, găsirea soluției optime este dificilă și nu întotdeauna garantată.

În alte cazuri, calculul mai multor valori (care trebuie să lucreze împreună pentru îndeplinirea unui scop comun și care sunt dependente una de alta) este o problemă foarte specifică. De multe ori, singura soluție este un algoritm „euristic" ghidat, probabil, de cineva cu experiență în domeniu.

Algoritmii genetici pot fi de folos în aceste cazuri datorită componentei aleatoare. Ne propunem să formulăm și să rezolvăm o problemă din această categorie [14].

2.1. Formularea problemei

Se dă sistemul de trafic feroviar din figura de mai jos. Există peroane (P_1, …, P_8), linii (L_1, …, L_4) și macazele situate la intersecțiile acestora (*interlocking*-urile I_1, …, I_6). Pe fiecare dintre aceste componente este permis să se afle maxim un tren la un moment dat. Evident, trenurile nu au voie să se ciocnească sub nici o formă. Există 3 rute predefinite pe care circulă trenuri (indicate în figură cu culori – galben: P_1, L_2, L_4, P_6, albastru: P_2, L_1, L_4, P_7, roșu: P_6, L_3, L_1, P_4).

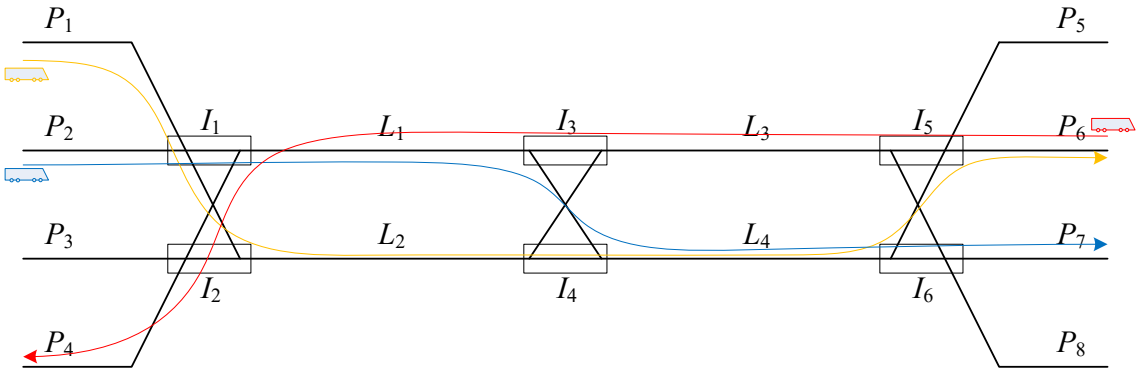


Figura 14. Sistemul de trafic feroviar

Timpii de parcurs pentru fiecare segment sunt indicați mai jos:

- Peroane (P_1, …, P_8) – 4 unități de timp
- Linii (L_1, …, L_4) – 6 unități de timp
- Interlocking-uri (I_1, …, I_6) – 1 unitate de timp

Se pune problema de a determina timpii de așteptare pentru fiecare tren, pe peroane și pe linii (exclus interlocking-uri), astfel încât fiecare tren să aștepte cât mai puțin și să nu existe ciocniri. Toate trenurile pornesc la momentul $t = 0$, dar pot exista întârzieri chiar la început [15],[16].

2.2. Rezolvarea problemei

Soluția problemei va fi reprezentată de un vector cu 12 componente, reprezentând timpii de așteptare (*delay*) pentru fiecare peron sau linie din cadrul rutei celor 3 trenuri. Structura cromozomului este deci:

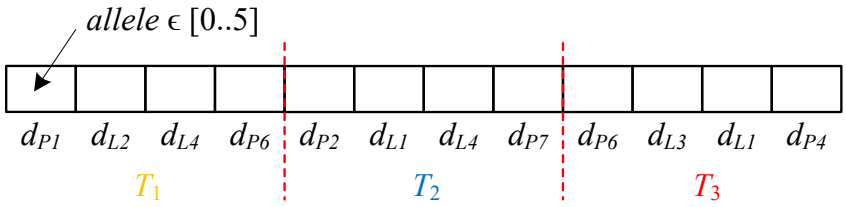


Figura 15. Structura cromozomului

Pentru această problemă, vom limita timpul de așteptare în intervalul [0, 5] unități de timp (genotipul) și vom realiza simularea sistemului cu un pas $\Delta t = 1$ unitate de timp. Alegerea întârzierilor nu garantează că trenurile nu se ciocnesc, de aceea trebuie luat și acest fapt în calculul performanței.

În caz că nu există ciocniri, performanța unui cromozom este calculată ca fiind timpul total de așteptare (d_{total}). Dacă există ciocniri, cromozomul este total neperformant și i se atribuie o valoare maximă:

if (*există_ciocniri*) **then** $J = MaxValue$ **else** $J = d_{total}$

2.3. Testare și concluzii

Găsirea soluțiilor valide și convergența

Dacă testăm algoritmul genetic cu funcția de performanță dată, vom observa că de multe ori nu generează nici o soluție fără ciocniri, adică cu performanța $J \neq MaxValue$. Aceasta se datorează faptului că toți cromozomii care generează ciocniri vor fi „la fel de neperformanți": există practic un salt al performanței care introduce o discontinuitate puternică a funcției de performanță în spațiul de căutare. Acest fapt poate conduce la imposibilitatea algoritmului de a ieși din zona de neperformanță. Pentru a evita acest lucru există mai multe variante:

- introducem suficientă varietate în populație de la început. În funcție de raportul dintre numărul de soluții cu ciocniri față de cele fără ciocniri, ar putea fi nevoie ca populația inițială să se apropie ca mărime de întreg spațiul de căutare, deci algoritmul să devină *brute force*.
- introducem o altă funcție de performanță care include, evident, timpul total de așteptare (d_{total}), dar și o componentă care penalizează puternic orice ciocnire. În formula de mai jos, $N_{ciocniri}$ este numărul de ciocniri în cadrul simulării, iar $f_{penalizare}$ este factorul scalar de penalizare.

$$J = d_{total} + f_{penalizare} \cdot N_{\text{ciocniri}}$$

Valoarea acestui factor va fi stabilită așa încât diferența procentuală între un cromozom care conține ciocniri și unul care nu conține ciocniri să fie de minim 80% (pentru a oferi operatorului de selecție probabilitate suficient de mare de a selecta cromozomul mai bun – o diferență de sub 50% nu ar garanta alegerea cromozomului mai performant). Cu ajutorul unui factor $f_{penalizare}$ suficient de mare, introducem practic intervale diferite de performanță (*clase de performanță*), care vizează comportamente diferite ale sistemului.

Timpul de așteptare d_{total} este în intervalul [0, 60], și avem nevoie de o diferență procentuală 80%. Rezultă deci $f_{penalizare} = 60 / (100\% - 80\%) = 300$.

Astfel, un cromozom cu 2 ciocniri va fi mai puțin performant decât unul cu o singură ciocnire (raport aproximativ de 50%), iar unul fără ciocniri va fi cu mult mai performant decât unul cu ciocniri (raport de 20%).

Lipsa soluțiilor valide

Dacă nu există o planificare a trenurilor conform condițiilor problemei, toți cromozomii vor avea cel puțin o ciocnire, deci $J > f_{penalizare}$. Din punctul de vedere al algoritmului, cea mai bună soluție va fi aceea cu J minim, deci cu cele mai puține ciocniri. Chiar dacă nu generează o soluție validă, algoritmul este totuși util pentru a oferi informații despre problemă: de exemplu, care tren este cel care generează ciocniri, sau care resursă este cea mai solicitată.

Oprirea algoritmului

Dacă, rulând algoritmul, descoperim că J devine mai mic decât $f_{penalizare}$ înseamnă că nu mai există ciocniri, astfel încât putem opri algoritmul după ce nu mai apar imbunatatiri semnificative. De obicei, se stabilește un prag de îmbunătățiri sub 5% – 10% pe parcursul a 10 generații consecutive.

3. Studiu de caz II: generarea rutelor trenurilor pentru sistemul feroviar

3.1. Formularea problemei

Considerând exemplul anterior, cu cele 3 rute prestabilite și timpii de așteptare pentru fiecare tren ca în figura de mai jos, se cere să se introducă un nou tren în plus (*charter*), care pornește din P_3 și ajunge pe oricare dintre peroanele din gara din dreapta. Evident, constrângerea împotriva ciocnirilor trebuie să se respecte cu strictețe și în acest caz. Așadar, trebuie calculată ruta și timpii de asteptare pe fiecare segment din rută pentru noul tren [16].

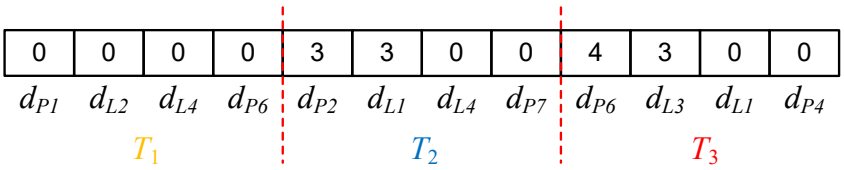


Figura 16. Întârzierile trenurilor planificate pentru fiecare segment

3.2. Rezolvarea problemei

Din cauza faptului că toți cromozomii trebuie să aibă lungime fixă, nu putem codifica ruta trenului în cromozom concatenând resursele care trebuie vizitate. Putem în schimb recurge la una dintre următoarele variante:

- gene care codifică poziția macazului din fiecare interlocking (în funcție de tipul macazului, gena va avea valori din mulțimea {0, 1} sau {0, 1, 2}) În acest caz, unele gene (interlocking-uri pe unde trenul nu circulă) nu vor avea efect asupra performanței. Este posibil ca unele soluții generate să nu fie valide, aspect analizat în problema precedentă.
- evaluarea dinainte a tuturor rutelor posibile, iar cromozomul va codifica index-ul rutei.
- observarea unor cazuri particulare ale problemei. De exemplu, putem observa că rutele posibile ale trenului sunt următoarele, unde perechile indicate în paranteze reprezintă variante posibile ale rutei:

  $P_3 \rightarrow (L_1, L_2) \rightarrow (L_3, L_4) \rightarrow (P_5, P_6, P_7, P_8)$

Observație: toate variantele prezentate mai sus pentru codificarea rutei în cromozom pot fi cu ușurință generalizate pentru un graf oarecare.

În cazul nostru, cromozomul va conține deci 3 gene care compun ruta, și încă 3 gene care vor reprezenta întârzierile pe fiecare dintre segmentele componente ale rutei (evident, nu este necesară o întârziere pe peronul de final). În figura de mai jos am indicat structura cromozomului și genotipul pentru fiecare genă în parte.

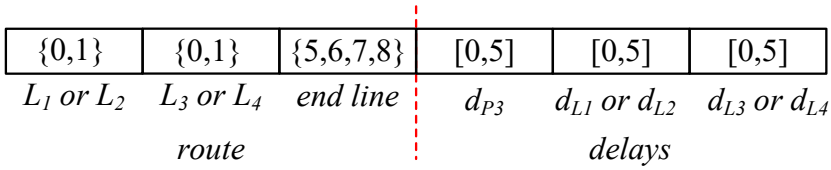


Figura 17. Structura cromozomului

Se observă că nu toate genele au același genotip, însă aceasta nu reprezintă un impediment pentru nici una din etapele buclei de evoluție a algoritmului genetic. Revăzând descrierea operatorilor de încrucișare (*crossover*) și mutație, observăm că:

- pentru încrucișare: fiecare genă își păstrează locul de ordine în cromozom: se schimbă câte o genă cu gena de pe poziția corespunzătoare a altui cromozom.
- pentru mutație: o genă ia o nouă valoare, aleasă din mulțimea genotipului acelei gene.

Funcția de performanță pentru această problemă are aceeași formă ca și în problema precedentă, penalizând puternic orice ciocnire a trenurilor și slab întârzierile.

4. Exerciții:

1. Implementați programul aferent studiului de caz I (determinarea întârzierilor). Testați programul mai întâi cu prima funcție de performanță, care nu penalizează diferențiat numărul de ciocniri. La a câta generație apare prima soluție fără ciocniri ?
2. Modificați funcția de performanță și folosiți $f_{penalizare} = 300$. Observați la ce generație scade brusc indexul J, semn că au fost eliminate ciocnirile care sunt penalizate puternic. Cum poate fi îmbunătățit algoritmul pentru a ajunge mai repede la acest rezultat ? Încercați diferite valori pentru $f_{penalizare}$ sau pentru funcția de performanță.
3. Implementați programul aferent studiului de caz II (determinarea rutei și a întârzierilor pentru un tren charter). Alegeți varianta indicată, observând cazul particular al problemei.
4. Testați programul de la exercițiul anterior inclusiv cu alte scenarii decât cel dat în Figura 16. Ce se întâmplă dacă scenariul folosit nu permite planificarea unui nou tren cu constrângerile date ?
5. Se dă un labirint sub forma unei matrice $N \times N$, cu valori 0 (pătrat liber) sau 1 (zid), si un prizonier pe unul din pătratele goale. Calculați cea mai scurtă rută de ieșire din labirint, folosind algoritmi genetici. Folosiți prima variantă indicată în capitolul 3.2.

Laborator 6

Introducere în programarea genetică

1. Obiectivele laboratorului

- Recapitularea conceptelor legate de programare genetică
- Însușirea unei metode de calcul a unei expresii matematice neliniare folosind programarea genetică
- Însușirea unei metode de determinare a structurii unui sistem ETPN folosind programarea genetică
- Exerciții

2. Recapitularea conceptelor de programare genetică

Pentru algoritmii genetici, cromozomul este un vector cu număr fix de gene, cu valori aparținând unui anumit genotip. *Programarea genetică* [17] este o tehnică din domeniul sistemelor evolutive, în care cromozomul are structură și mărime variabile, fiind reprezentat de obicei sub formă de arbore, așa cum se vede în exemplul de mai jos. Astfel, sunt favorizate limbajele care folosesc structuri de arbori în mod natural (de exemplu, Lisp sau alte limbaje funcționale).

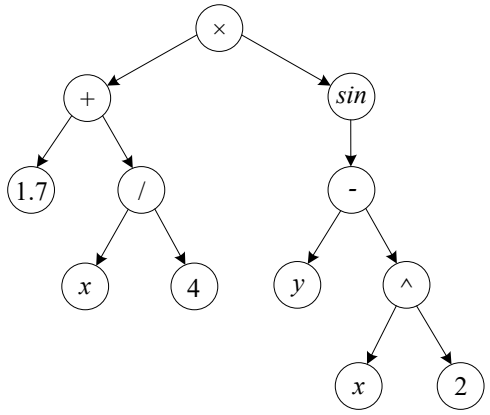


Figura 18. Exemplu de cromozom sub formă de arbore

Nodurile interne (neterminale) conțin operatori sau funcții, deci genotipul aferent este de exemplu: G_{nod} = {+, -, ×, /, ^, √, *sin*, *cos*, *log*, etc.} sau, folosind operatori logici: G_{nod}' = {*if-then-*(*else)*, *or, not*, *and*, >, ≤, etc.}. Aici, trebuie ținut cont că numărul de copii să fie potrivit cu natura funcției, de exemplu doi copii pentru +, și unul pentru *sin*.

Nodurile terminale (frunze) conțin valori (operanzi): fie constante, fie variabile de intrare: $G_{terminal} = \{x, y\} + R_+$.

Așadar, un cromozom poate codifica programe software, rezultatul algoritmului fiind un program care rezolvă o anumită sarcină (*task*), conform funcției de performanță.

Operatorii genetici trebuie adaptați la structura cromozomului: *încrucișarea* se face prin tăierea unui subarbore și înlocuirea lui cu un subarbore de la alt cromozom, iar pentru *mutația* unui nod trebuie aleasă o valoare din mulțimea genotip corespunzătoare. Încrucișarea modifică structura cromozomului, deci numărul total de noduri ar putea să crească excesiv, devenind nepractic din punct de vedere computațional.

Se pot introduce totuși diferite constrângeri, de exemplu o adâncime maximă de ordinul $d_{max} = 5$ nivele sau limitarea numărului total de noduri.

Pentru a evalua cromozomul sub formă de arbore, se face mai întâi transformarea arborelui în individul corespunzător, prin *mapping*. Individul este în acest caz, expresia echivalentă sub forma aleasă (expresie matematică, program, expresie Lisp, etc). Construirea expresiei se face cel mai ușor parcurgând arborele în mod recursiv. De exemplu, rezultatul *mapping*-ului pentru cromozomul dat mai sus este expresia matematică $E(x,y) = (1.7 + x / 4) \times sin(y - x^2)$.

Evalurea efectivă a expresiei echivalente (individul) se face într-un mod adecvat, în funcție de specificul aplicației. De exemplu, se pot înlocui operanzii x și y cu valori numerice, se poate evalua structura arborelui sau performanța computațională a expresiei.

Observație

Pentru o viteză computațională superioară, se poate opta pentru implementarea tuturor funcțiilor (inițializare, operatori genetici, evaluare) direct pentru reprezentarea sub formă Lisp a expresiei. Aceasta evită alocarea de memorie (foarte costisitoare temporal) pentru crearea fiecărui nod și, în schimb, efectuează toate procesările pe șiruri de caractere (*string*-uri).

3. Studiu de caz I: determinarea unei expresii matematice / logice

Problema generală este aceea de a determina cea mai bună expresie matematică sau logică, care produce niște rezultatele specificate. Expresia cerută are una sau mai multe variabile de intrare care modifică valoarea/valorile de ieșire. În exemplul următor se dă un tabel de valori (*engl*. *lookup table*) și se dorește determinarea unei expresii logice (program) cu intrarea x care returnează valorile date y.

3.1. Formularea problemei

Se dă tabelul de valori de mai jos:

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
y	0	0	-3	0	0	0	1	2	3	4	5

Se cere să se determine o expresie logică (program) care primește ca intrare o valoare x și generează ca rezultat valoarea y corespunzătoare. Domeniul pentru valoarea de intrare este $x \in [-5, 5]$. Se consideră ca o expresie este cu atât mai performantă cu cât numărul total de noduri este mai mic.

3.2. Rezolvarea problemei

Pentru că problema vizează determinarea unei expresii logice, vom considera mulțimile genotip următoare:

G_{nod} = {*if-then-else, or, and, >, ==*} și $G_{terminal}$ = {x, -5, -4,…, 0, …, 5}

Se observă că din G_{nod} am eliminat funcția ≥, deoarece aceasta poate fi obținută prin operația (> *or* ==), toate cele 3 funcții fiind disponibile. În alte cazuri similare, se poate elimina de exemplu *tg* = *sin* / *cos* (dacă *sin* și *cos* sunt disponibile). Nu există o regulă clară referitor la ce funcții sau valori terminale trebuie să fie incluse în mulțimile genotip, aceasta ține mai mult de problemă și experiența programatorului.

Vom considera adâncimea maximă a arborelui d_{max} = 5, iar funcția de evaluare:

$$J(chr) = \sum_{x=-5}^{5} \left(y(x) - eval\_chr(x) \right) + nr\_nodes$$

, unde *eval_chr*(*x*) reprezintă rezultatul evaluării cromozomului *chr* pentru valoarea lui *x* dată ($x \in [-5, 5]$), și *nr_nodes* reprezintă penalizarea aferentă numărului total de noduri. La nevoie, fiecare dintre aceste două contribuții pot fi scalate cu un factor proporțional.

3.3. Implementare

În pachetul *jgap* există programul *SimpleExample* care implementează o problemă similară. Din acest motiv, nu reproducem aici codul implementat. Este necesară modificarea doar a mulțimilor genotip și a funcției de performanță.

4. Studiu de caz II: determinarea unei expresii TPNL

În cadrul acestui exemplu, vom folosi programarea genetică pentru a determina structura unei rețele ETPN (Enhanced Time Petri Net) pe baza expresiei TPNL (Time Petri Net Language) echivalente [18], [19].

4.1. Rețele ETPN și limbajul TPNL

Rețelele ETPN sunt rețele Petri temporizate cu porturi de intrare (*reacție*) și ieșire (*control*) care permit interconectarea cu alte rețele similare. Porturile sunt, de fapt, locații care sunt partajate (folosite în comun) cu celelalte rețele conectate.

Notația *TPNL* este o variantă compactă de reprezentare a unei rețele ETPN. În acest limbaj, o tranziție este notată $t_i[r_j, c_k]$, unde:

- r_j reprezintă un *canal de reacție* (condiția de activare a tranziției), adică un arc de la un port de intrare (Pr_j) sau o temporizare τ, care poate fi și τ=0.
- c_k reprezintă un *canal de control* (acțiunea facută de tranziție), adică un arc spre un port de ieșire (Pc_k). Lipsa arcului de control se notează cu φ.

Există mai multe variante de execuție a tranzițiilor sau a secvențelor de tranziții, care pot fi descrise identic în limbajul TPNL:

Operand	Semnificație	Notație TPNL
*	execuție secvențială	exp_1*exp_2
+	execuție alternativă	exp_1 + exp_2
&	execuție concurentă	exp_1 & exp_2
#	execuție repetitivă	exp_1 # exp_2

Expresiile TPNL descriu rețele ETPN echivalente, într-un limbaj compact. Prin concatenarea succesivă a mai multe expresii TPNL, se pot obține rețele ETPN complexe. Se oferă spre exemplu expresia următoare:

$$\sigma_1 = ((t_1[r_1,\varphi] * t_2[4,c_1]) \,\&\, (t_3[r_2,\varphi] * (t_4[3,c_2] + (t_5[r3,\varphi] * t_6[1,c_3]))) \,\#\, t_7[10,\varphi]$$

Conversia din expresie TPNL în reprezentarea sub formă de arbore se rezolvă recursiv și este imediată. Devine evident, deci, că problema de a genera structura ETPN potrivită pentru un anumit scop (de exemplu de control al unui sistem) poate fi abordată prin programare genetică.

4.2. Formularea problemei

În figura de mai jos este prezentat un model ETPN pentru trecerea la nivel cu calea ferată. Rețeaua are porturi de intrare (P_{c1}, P_{c2}) prin care este controlată poziția barierei și porturi de ieșire (P_{r1}, P_{r2}) care sunt activate de apropierea, respectiv de îndepartarea trenului. Trenul parcurge un segment în $\tau = 5$ unități de timp, iar trecerea vehiculelor peste calea ferată este controlată de barieră.

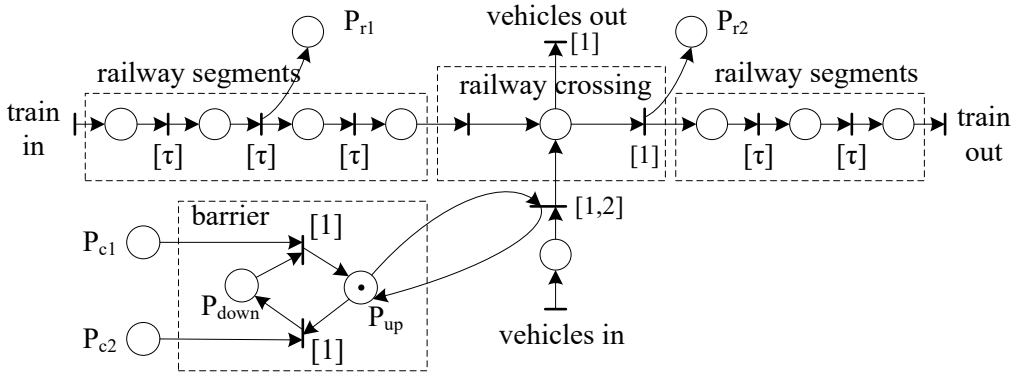


Figura 19. Model ETPN al trecerii la nivel cu calea ferată

Se cere să se construiască un sistem de control ETPN care să controleze trecerea la nivel cu calea ferată în siguranță (nu se permit ciocniri) și în mod cât mai eficient (numărul de vehicule care pot trece într-o perioadă de simulare T_{sim} să fie maxim). Se va considera un flux constant de vehicule (1 vehicul / secundă) și două sau mai multe trenuri succesive la intervale date. O problemă similară, care poate fi formulată și rezolvată în același mod, este controlul trecerii vehiculelor printr-o intersecție [20].

4.3. Rezolvarea problemei

Vom considera mulțimile genotip următoare:

$G_{nod} = \{*, +, \&, \#\}$, $G_{reacție} = \{r_1, r_2, 0, \ldots, 5\}$, $G_{control} = \{c_1, c_2, \varphi\}$

Se observă că în acest caz, $G_{terminal} = G_{reacție} + G_{control}$, cu mențiunea că pentru un nod terminal (tranziție) se vor alege doi parametri: canalul de reacție (din $G_{reacție}$), și canalul de control (din $G_{control}$). La operația de mutație (în cazul unui nod terminal) se va considera posibilitatea mutației numai a reacției sau numai a controlului. În consecință, trebuie rescris operatorul de mutație folosit de algoritm cu unul personalizat.

Funcția de performanță va lua în considerare mai multe contribuții: existența accidentelor (un tren și un vehicul se află simultan în intersecția dată) și numărul total de vehicule care reușesc să treacă. Pentru fiecare dintre aceste contribuții considerăm un factor de scalare potrivit. De asemenea, un sistem de control mai simplu este considerat mai bun decât unul complex, deci se va penaliza și numărul total de noduri.

$J(chr) = c_1 \cdot nr\_accidents - c_2 \cdot nr\_vehicles\_pass + nr\_nodes$

4.4. Implementare

Pentru un arbore de adâncime d_{max} = 5, mărimea spațiului de căutare în această configurație devine:

$$|S| > |G_{terminal}|^{N_{terminal}}(d_{max}) \cdot |G_{nod}|^{N_{nod}}(d_{max})$$

, unde ^ este operația de ridicare la putere, iar $N_{terminal}$ și N_{nod} sunt numărul de noduri terminale respectiv neterminale pentru adâncimea dată a arborelui. În condițiile problemei,

$$|S| > (7 \cdot 3)^{2^5} \cdot 4^{(2^3+2^2+2^1+2^0)} \approx 2.19 \cdot 10^{51}$$

Din cauza spațiului de căutare extrem de vast, algoritmul standard care tratează fiecare nod ca obiect (care trebuie alocat/dealocat la fiecare operație) devine foarte încet și ineficient în găsirea unei soluții utile. În consecință, se optează pentru implementarea tuturor operațiilor (crossover, mutație, evaluare, generare de cromozom inițial) direct pe expresia de tip TPNL a cromozomului, folosind șiruri de caractere. În acest fel, se reduce considerabil nevoia de a instanția obiecte noi. În limbajul Java, se pot folosi obiecte de tip `StringBuilder`, care permit operații de editare direct pe șirul de caractere. Algoritmul implementat este disponibil la adresa: https://github.com/cuibus/GPonString, împreună cu un exemplu de utilizare.

4.5. Testare

Vom considera adâncimea maximă a arborelui d_{max} = 5, iar pentru evaluarea unui cromozom (sistem de control) vom realiza o simulare cu T_{sim} = 50 secunde. Vehiculele vor intra în sistem cu perioada dată (1 vehicul / secundă), și vom introduce jetoane reprezentând trenuri în partea stângă la momentele T_1 = 5s, T_2 = 20s, T_3 = 25s. Sistemul se va simula cu ajutorul rețelei ETPN descrisă în figura de mai sus.

5. Exerciții

1. Folosind exemplul *SimpleExampleGP* din pachetul *jgap* ca punct de pornire, construiți programul pentru studiul de caz I. Pentru aceasta, setați mulțimile genotip G_{nod} si $G_{terminal}$ și implementați funcția de performanță așa cum sunt acestea descrise în capitolul 3.2. Rulați programul și notați expresia rezultată.
2. Studiați influența parametrilor cromozomului: adâncimea maximă, numărul total de noduri, factori de scalare, etc. asupra performanței de a găsi soluția optimă.
3. Construiți grafic rețeaua ETPN pentru expresia lui σ_1 dată în capitolul 4.1. Pentru expresia TPNL dată ca `string`, construiți operatorul de mutație așa cum este el descris în capitolul 4.3. Se vor folosi probabilități configurabile pentru mutația unui operator, nod terminal – reacție sau nod terminal – control.
4. Construiți programul pentru studiul de caz II. Se va folosi simulatorul pentru sistemul dat disponibil la următoarea adresă: https://github.com/cuibus/RailCrossingSimulator și pachetul de programare genetică care lucrează pe șiruri de caractere https://github.com/cuibus/GPonString. Folosiți operatorul de mutație implementat la exercițiul 3 sau cel deja disponibil. Implementați funcția de

performanță și scenariul de simulare. Rulați programul și interpretați fizic sistemul de control generat.

5. Construiți manual, pe hârtie, cel mai bun sistem de control ETPN care respectă specificațiile problemei pentru studiu de caz II. Calculați performanța acestuia și opriți algoritmul dacă ajunge la o diferență de performanță de sub 5% față de cel mai bun sistem construit. Cât timp rulează algoritmul acum ? Poate fi îmbunătățit sistemul ?

Laborator 7
Optimizarea multiobiectiv

1. Obiectivele laboratorului

- Recapitularea conceptelor legate de optimizarea multiobiectiv: frontul Pareto
- Însușirea unei metode de optimizare multiobiectiv a unei funcții matematice
- Analiza frontului Pareto și condiția de identificare a frontului
- Însușirea unei metode de stabilire a parametrilor unui sistem de control multiobiectiv, în varianta fuzzy

2. Recapitularea conceptelor legate de optimizarea multiobiectiv

Problemele de optimizare vizează uneori mai multe obiective care sunt în conflict, adică ridicarea performanței aferente unui obiectiv poate duce la scăderea performanței altor obiective (de exemplu: suprareglaj ↔ perioadă de stabilizare). În aceste cazuri se alege o variantă de compromis, iar soluția finală va satisface toate obiectivele parțial.

O soluție posibilă din spațiul de căutare se numește:

- *Pareto-dominată* dacă există o altă soluție posibilă care îmbunătățește unele obiective și nu înrăutățește nici un alt obiectiv.
- *Pareto-optimală* dacă nici un obiectiv nu poate fi îmbunătățit fără a înrăutăți alte obiective.

Frontul Pareto este zona formată din toate soluțiile Pareto-optimale și delimitează spațiul de căutare acolo unde un obiectiv nu poate fi îmbunătățit fără a afecta negativ un alt obiectiv. Frontul Pareto este o linie (dacă există 2 obiective), o suprafață (3 obiective), sau o hiper-suprafață. Toate soluțiile Pareto-dominate se află de aceeași parte a frontului Pareto.

Se consideră că toate soluțiile de pe frontul Pareto sunt în mod egal performante. Alegerea finală a unei soluții implică preferința subiectivă a utilizatorului. De exemplu, se poate opta pentru acea soluție care maximizează (sau minimizează) suma obiectivelor.

La optimizarea multiobiectiv, funcția de performanță este multidimensională cu dimensiunea n: $J : S \rightarrow R^n$, unde $R^n = R \times R \times ... \times R$ este produsul cartezian al mulțimii reale R de n ori. Putem considera că este vorba, practic, de n funcții obiectiv.

Selecția cromozomilor trebuie să păstreze varietatea populației cu privire la îndeplinirea tuturor obiectivelor [21], deci este necesară implementarea unui operator de selecție specializat:

- folosirea succesivă a operatorului clasic de selecție, pentru fiecare obiectiv, pentru a genera subpopulații. Generația următoare este compusă din toate aceste subpopulații.
- selecția folosind un indice de performanță care este o sumă ponderată a obiectivelor,

- înlocuirea progresivă a fiecărei soluții Pareto-dominate cu una dintre soluțiile Pareto-optimale care o domină.

În figura de mai jos, există 2 obiective f_1 și f_2 care trebuie minimizate simultan. Pentru fiecare soluție Pareto-dominată există cel puțin o soluție Pareto-optimală care îmbunătățește cel puțin un obiectiv. În practică, se evaluează mai întâi frontul Pareto, iar apoi se alege în mod subiectiv (empiric) una dintre soluțiile respective.

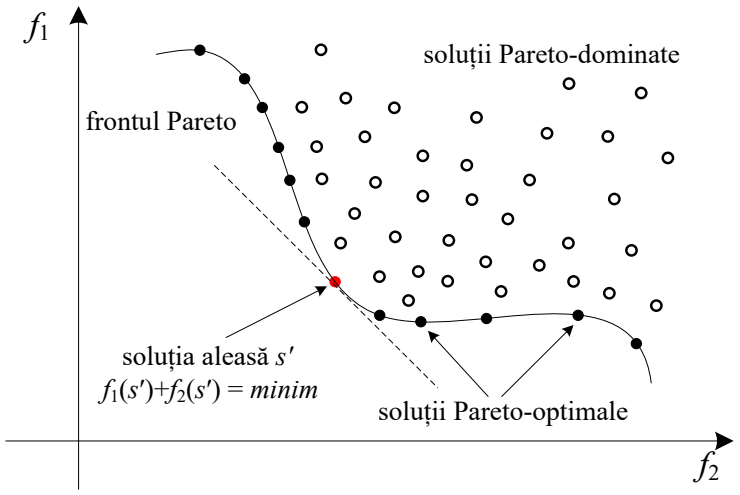


Figura 20. Optimizare multiobiectiv. Frontul Pareto

3. Studiu de caz I: optimizarea multiobiectiv a unei funcții matematice

Problema generală este aceea de a găsi valoarea numerică pentru argumentele x_1, x_2, ..., x_n astfel încât funcțiile obiectiv f_1, f_2, ... f_m: $R^n \rightarrow R$ să fie minimizate (sau maximizate) simultan. Domeniul de definiție al funcțiilor poate fi constrâns, dar este necesar să fie identic pentru toate funcțiile. O notație echivalentă este să considerăm o singură funcție obiectiv $f:R^n \rightarrow R^m$, cu m valori de ieșire. Se cere identificarea frontului Pareto și apoi selectarea unei soluții care satisface o condiție dată.

În exemplul următor, funcțiile obiectiv au o singură variabilă, iar selecția soluției de pe frontul Pareto se va face maximizând o combinație liniară a funcțiilor obiectiv.

3.1. Formularea problemei

Se dă o funcție matematică având o singură variabilă ca intrare notată cu x și două valori de ieșire, notate f_1 si f_2. Funcția f este deci un tuplu de 2 funcții obiectiv:

$f: [-\pi/2, \pi] \rightarrow R \times R, f(x) = (f_1 ; f_2) = (\sin(x) ; \cos(x))$

Se cere să se afle valoarea lui x pentru care f_1 și f_2 au valori cât mai mari.

3.2. Rezolvarea problemei

Se vede imediat că maximul pentru funcțiile f_1 și f_2 se atinge pentru $x_1 = 0$, respectiv $x_2 = \pi/2$, prin urmare f_1 și f_2 nu pot fi maximizate simultan cu aceeași valoare a lui x. Spunem deci că cele două obiective sunt în conflict.

Pentru vizualizarea problemei, se dă reprezentarea grafică a celor două funcții mai jos. Se observă că în afara intervalului $[0, \pi/2]$ ambele obiective pot fi îmbunătățite

simultan, deci ne așteptăm ca soluția optimă x să se găsească în interiorul acestui interval, iar frontul Pareto să fie format din evaluarea lui f_1 și f_2 pentru puncte x, unde $x \in [0, \pi/2]$.

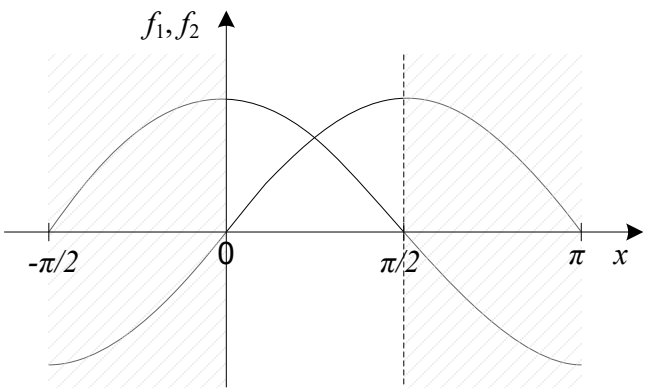


Figura 21. Funcțiile obiectiv pentru studiul de caz I

Pentru a reprezenta grafic frontul Pareto recurgem la expresia lui f_2 în funcție de f_1, și anume: $f_2(f_1) = \cos(\arcsin(f_1))$. Reprezentarea grafică este în figura următoare. În mod normal, la problemele unde rezolvarea nu este directă, frontul Pareto este determinat prin rularea algoritmului genetic până când nu mai apar îmbunătățiri semnificative. Populația din ultima generație conține indivizi care se află, preponderent, pe frontul Pareto.

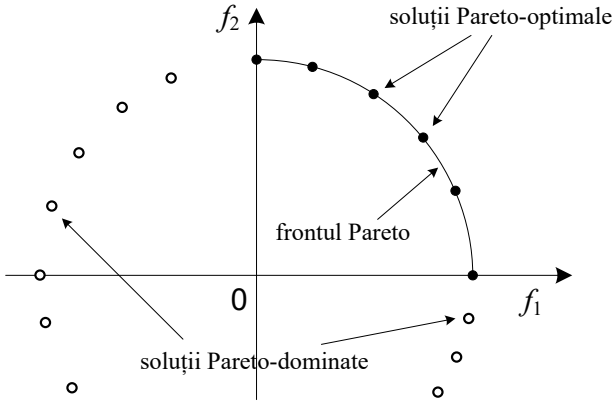


Figura 22. Frontul Pareto pentru studiul de caz I

Pentru alegerea unei soluții dintre cele Pareto-optimale, vom recurge de această dată la suma ponderată a celor două obiective: vom alege x pentru care $f = 2 \cdot f_1(x) + 3 \cdot f_2(x)$ este maximă. În general, alegerea soluției finale de pe frontul Pareto se poate face fie după reprezentarea lor grafică (alegere subiectivă), fie utilizând un algoritm genetic simplu, pentru care spațiul de căutare va fi compus din soluțiile Pareto-optimale.

3.3. Implementare

Modul de implementare nu diferă substanțial față de algoritmul genetic standard. Printre exemplele descărcare odată cu pachetul *jgap*, în directorul *multiobjective* există un

exemplu *MultiObjectiveExample*, care oferă această implementare și pe care nu îl mai reproducem aici.

În pachetul *jgap* există niște clase cu metode definite pentru rezolvarea problemelor de optimizare multiobiectiv:

- `BulkFitnessFunction`, cu metoda `void evaluate(Population a_subject)`, cu ajutorul căreia se evaluează toate obiectivele pentru întreaga populație. Performanța de îndeplinire a fiecărui obiectiv se setează în obiectul de tip `Chromosome`, cu metoda `setMultiObjectives(List a_values)`. La sfârșitul evoluției programului, populația finală va conține indivizi de pe frontul Pareto.
- Selectorul `BestCromosomesSelector` este cel folosit pentru păstrarea în populație a indivizilor performanți la fiecare dintre obiective.
- `FitnessEvaluator`, cu metoda `isFitter(…)`, folosită pentru alegerea soluției finale de pe frontul Pareto determinat, prin comparația cromozomilor relativ la performanțele de îndeplinire ale fiecărui obiectiv. Semnătura acestei metode este: `boolean isFitter(IChromosome a_chrom1, IChromosome a_chrom2)`

3.4. Testare și concluzii

Frontul Pareto

Dacă se rulează programul pentru 20 de generații, în populația finală se vor regăsi valorile lui x care fac parte din frontul Pareto. Încercarea de a adăuga pe o hartă carteziană puncte cu coordonatele $(f_1(x), f_2(x))$ va rezulta într-un grafic cum este cel din figura 24. Alegerea soluției preferate se poate face acum în mai multe moduri:

- subiectiv, direct de pe grafic.
- *brute force*, calculând valoarea sumei ponderate $f = 2 \cdot f_1(x) + 3 \cdot f_2(x)$ pentru fiecare punct de pe frontul Pareto. O alternativă la brute force este căutarea binară, după ordonarea soluțiilor.
- dacă frontul Pareto conține prea multe soluții, se poate folosi un algoritm genetic simplu în loc de abordarea *brute force*, cu funcția de performanță f dată.

Convergența

Dacă lăsăm algoritmul să ruleze mai departe, observăm că, după ce populația s-a stabilizat în jurul frontului Pareto, nu există o convergență standard în sensul în care cea mai bună soluție să nu se mai modifice. Populația conține indivizi de pe frontul Pareto și va baleia în continuare zona respectivă, găsind cu fiecare generație alte soluții diferite de pe frontul Pareto. Nu se poate vorbi despre noțiunea de convergență standard pentru că nu există o soluție finală care să fie cea mai bună.

Identificarea momentului când algoritmul ajunge la frontul Pareto se poate face pe baza definiției Pareto-optimalității: nici o soluție găsită nu mai poate fi îmbunătățită în ceea ce privește *toate* obiectivele. Este necesar deci să verificăm dacă există vreun individ din generația anterioară față de care un individ din generația actuală îmbunătățește toate obiectivele.

4. Studiu de caz II: generarea unui sistem de control multiobiectiv

4.1. Formularea problemei

Se dă un sistem format dintr-un lac de acumulare cu două turbine generatoare de energie electrică, G_1 si G_2, controlate prin debitele s1 și s2. Acestea au expresiile: $s_1=\alpha \cdot c_1$, $s_2=\alpha \cdot c_2$, unde c_1 și c_2 sunt mărimile de control, iar α este constantă. Aportul de apă în lac, u, se consideră măsurabil.

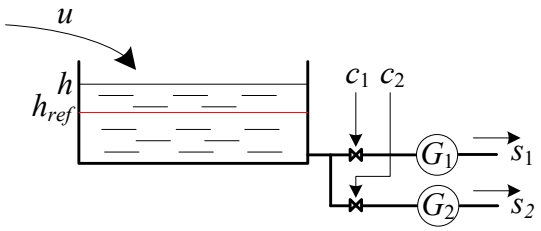


Figura 23. Lac de acumulare cu două generatoare

Sistemul are următorul model discret pentru h – înălțimea apei și P_i – puterea generată de generatorul i:

$$h(k+1) = h(k) + A \cdot \left(u(k) - s_1(k) - s_2(k)\right)$$

$$P_i(k) = K_i \cdot h(k) \cdot s_i(k)$$

, unde A și K_i sunt constante (aria bazinului, respectiv o constantă a generatorului) [22],[23]. Se consideră că generatoarele sunt diferite ca și specificații ($K_1 > K_2$).

Se cere să se calculeze parametrii unui sistem de control fuzzy pentru care nivelul apei se menține într-un interval în jurul referinței h_{ref} și energia totală generată este maximă.

4.2. Rezolvarea problemei

Pentru înălțimea coloanei de apă din lac, vom considera nivelele fuzzy descrise în laboratorul 4: *NL*, *NM*, *ZR*, *PM*, *PL*, cu mențiunea că *ZR* corespunde lui h_{ref}. Aceleași valori fuzzy vor fi folosite pentru debitul de intrare u și pentru comenzile c_1 și c_2. Evident, fiecare dintre aceste mărimi poate folosi funcții de apartenență diferite.

Regulile de control fuzzy sunt de forma dată mai jos. Sunt necesare 25 de reguli de control, deci soluția problemei va fi un tabel de dimensiune 5×5 cu perechi de reguli (c_1, c_2).

IF (u IS U AND h IS H) THEN (c_1 IS C_1 AND c_2 IS C_2)

Cromozomul codifică tabelul de reguli fuzzy, deci conține 25 de gene cu valori întregi în intervalul [1, 5], așa cum s-a folosit în laboratorul 4.

Performanța sistemului vizează două obiective care sunt în conflict: minimizarea erorii de nivel $\Delta h = |h_{ref} - h|$ și maximizarea energiei totale generate, ambele pe un orizont de optimizare T. Prin urmare, funcțiile obiectiv au expresiile:

$J_{\Delta h} = \sum_{k=0}^{T} \left| h_{ref}(k) - h(k) \right|$, unde $J_{\Delta h}$ (suma erorilor nivelului) trebuie minimizată

$J_e = E_{tot} = \sum_{k=0}^{T} \left(P_1(k) + P_2(k)\right) \cdot \Delta t_k$, unde J_e (energia totală generată) trebuie maximizată. În acest caz, $\Delta t_k = 1$ reprezintă perioada de eșantionare.

Pentru a evalua performanța unui cromozom, este necesară simularea sistemului pe întreg orizontul de optimizare și memorarea valorilor nivelului h și a puterii generate P_1 si P_2. De menționat că $J_{\Delta h}$ trebuie minimizată, iar J_e trebuie maximizată, deci este nevoie să se inveseze semnul la una dintre ele.

Funcția globală de evaluare a performanței unui sistem de control este: $J = J_e - \mu \cdot J_{\Delta h}$, unde μ este o pondere introdusă cu scopul de a echilibra sensibilitatea lui J în raport cu $J_{\Delta h}$. Această funcție va fi folosită în final pentru alegerea unei soluții de pe frontul Pareto.

5. Exerciții

1. Folosind exemplul *MultiObjectiveExample* din pachetul *jgap* ca punct de pornire, construiți programul pentru studiul de caz I. Exemplul din în pachetul *jgap* realizează optimizarea multiobiectiv pentru o funcția de performanță: $f = (f_1, f_2) = (x^2; (x-2)^2)$. Este nevoie deci, să modificați funcția de performanță și funcția care alege soluția finală.
2. După găsirea frontului Pareto, reprezentați grafic soluțiile găsite așa cum este indicat în capitolul 3.4. Rulați algoritmul pentru încă 30 de generații, refăcând de fiecare dată graficul, în locul celui anterior. Ce observați din animația rezultată ?
3. Folosind același exemplu, implementați programul pentru studiul de caz II, în varianta de control fuzzy. Alegeți valori potrivite pentru constantele date și pentru valorile fuzzy astfel încât problema să aibă sens.
4. Pentru studiul de caz II, reprezentați grafic frontul Pareto prin adăugarea de puncte aferente fiecărei soluții pe o reprezentare carteziană ($J_{\Delta h}$, J_e). Analizați forma, capetele și caracteristicile acestuia și înțelegeți legătura dintre parametrii de control fuzzy și frontul Pareto.

Alte probleme propuse

1. Probleme de optim în matematică

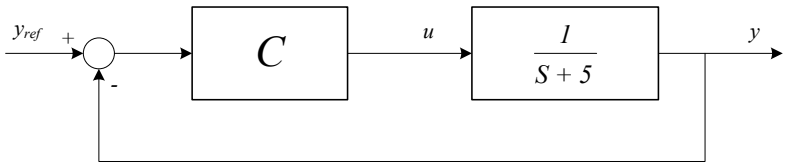
- Aflați soluția ecuației $f(x) = 5$, unde $f : [-10, 10] \rightarrow R$, $f(x) = x^4 - 3 \cdot x^2 + 2 \cdot x$, cu o precizie $\delta = 0.005$.
- Aflați maximul funcției $f : R \times R \rightarrow R$, $f(x, y) = x \cdot 1.5^{-x^2 - y^2}$ unde $x, y \in [-5, 5]$, cu o precizie $\delta = 0.01$ pentru fiecare variabilă. Verificați rezultatul.
- Găsiți cea mai bună funcție polinomială de grad maxim 4 care aproximează funcția exponențială e^x pe intervalul $[0, e]$.
- Se dă expresia $f(x) = a \cdot sin(b \cdot x) + c \cdot cos(d \cdot x)$. Aflați valorile reale a,b,c,d pentru care derivata $f'(x) < f(x)$ pe tot intervalul $[-5, 5]$.

2. Modificarea unei distribuții de probabilitate

- Un examen teoretic constă din răspunsul la 21 de întrebări grilă, fiecare cu o anumită pondere (punctaj). Se dă o matrice cu 100×21 valori `boolean`, reprezentând rezultatele la evaluarea a 100 de participanți. Se cere să se determine ponderea fiecărei întrebări, astfel încât cât mai mulți participanți să obțină punctaj total peste 15, iar diferența între ponderi să nu depășească 50%.
  Alternativ, se cere găsirea ponderilor astfel încât notele finale să respecte cât mai bine o distribuție gaussiană.
- Se dă o funcție *rand_0_pi*( ) care returnează un număr aleator în intervalul $[0, \pi]$, cu distribuția de probabilitate $P(x) = sin(x)$. Găsiți o funcție polinomială de grad maxim 5 care să transforme distribuția dată într-o distribuție cât mai uniformă.

3. Calculul parametrilor unui controler PID

Se dă sistemul de gradul I din figura de mai jos. Calculați parametrii sistemului de control *C*, astfel încât răspunsul sistemului la treaptă să fie cât mai bun. Se cere să calculați parametrii controlerului în varianta P (proporțional) și PI (proporțional-integrativ). Se va considera un orizont de optimizare adecvat. *Sugestie*: discretizați sistemul și considerați un orizont de optimizare egal cu constanta de timp a sistemului.



4. Acoperire optimă pentru o suprafață

- Se dă o suprafață rectangulară de dimensiune fixă. Există *n* routere, fiecare cu o anumită rază de acoperire. Unde trebuie ele plasate ca să acopere cât mai bine zona

dată ? Zonele de lângă ferestre au prioritate pentru că acolo se vor amplasa birouri. Ne interesează acoperirea unei suprafețe cât mai mari.

- Într-o fabrică de îmbrăcăminte se taie cu o ștanță 20 de forme date dintr-un material textil cu lățimea dată (rolă). Se cere optimizarea tăieturilor în așa fel încât lungimea consumată din material să fie cât mai mică.

5. Control fuzzy ierarhizat pentru un sistem de lacuri de acumulare

În cazul controlului fuzzy pentru un sistem cu un număr mare de intrări și ieșiri, rezultă un număr imens de reguli de control, ceea ce face impractică folosirea algoritmilor genetici pentru găsirea lor. Exemplul de mai jos reprezintă modelul unui ansamblu de 4 lacuri, deci în total 8 intrări (debit de intrare u_i și nivelul apei l_i pentru fiecare lac) și 4 ieșiri d_i. Ar fi nevoie deci de $N = 5^8 \cdot 4 = 1\ 562\ 500$ reguli pentru controlul fuzzy cu un singur bloc FLRS.

Observăm în schimb că putem distribui controlul în mai multe unități, fiecare responsabilă de un singur lac. Mai mult, mărimile de intrare d_1 și u_2 pentru lacul 2, la fel ca d_2, d_3 și u_4 pentru lacul 4 le putem însuma, nu este necesar să le considerăm separat. Ajungem deci la o *structură cu blocuri fuzzy ierarhizate*, cu un număr total de $N = 5^2 \cdot 4 = 100$ de reguli de control, mult mai fezabil pentru implementarea cu algoritmi genetici.

Implementați un program Java care stabilește regulile de control pentru acest sistem, cu scopul de a menține nivelul apei în fiecare lac la o referință dată, indiferent de mărimile de intrare u_1, u_2, u_3, u_4 (măsurabile, dar necontrolabile).

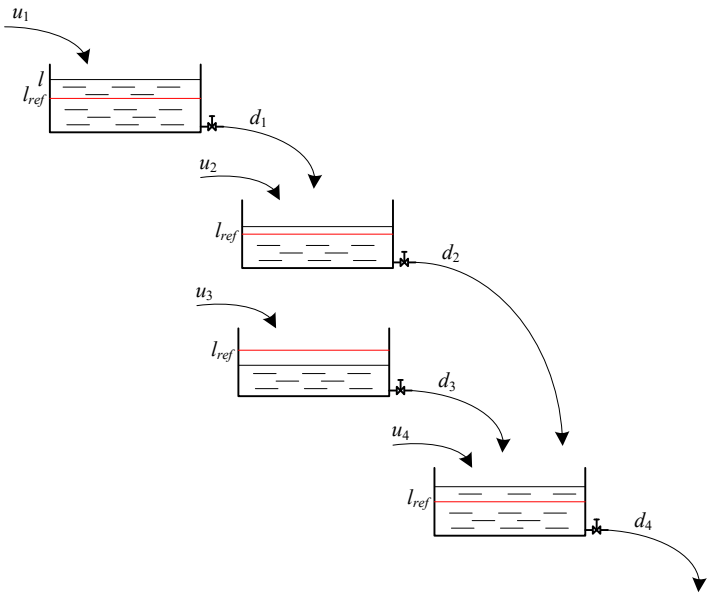


Figura 24. Modelul unui ansamblu de 4 lacuri

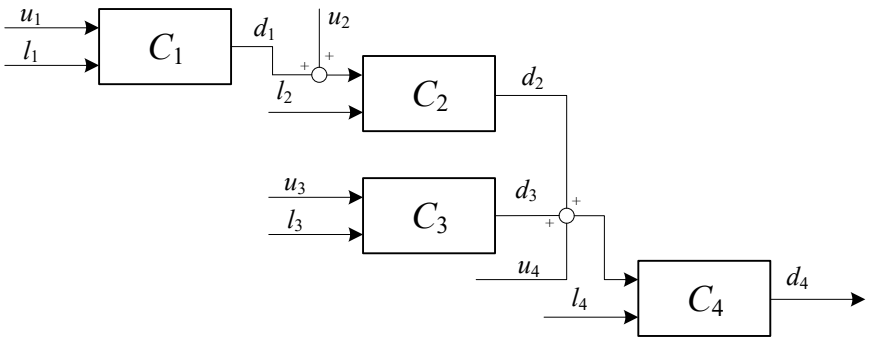


Figura 25. Control fuzzy ierarhizat

6. Problema comis-voiajorului

- Se dă un graf neorientat cu N orașe, cu costurile de deplasare de la un oraș la altul cunoscute. Un comis-voiajor are N pachete cu greutățile g_1, g_2, ..., g_n și trebuie să livreze fiecare pachet în orașul corespunzător. Calculați ruta optimă a comis-voiajorului astfel încât distanța totală parcursă să fie cât mai mică, dar să se și evite deplasarea îndelungată cu pachete grele.
- Într-un oraș există m magazine, fiecare având un set de produse S_i, i=1,..,m, cu prețurile aferente. Lista de produse nu este comună, dar se suprapune parțial la diferite magazine. Un cetațean are nevoie de anumite produse pe care le va cumpăra. Trebuie determinat de la care magazin va cumpăra fiecare produs astfel încât costul total să fie cât mai mic, dar și numărul de magazine vizitate să fie cât mai mic.

7. Determinarea rutelor într-un sistem

- Se dă o rețea de n×n intersecții în cruce, prin care vehiculele pot circula virând la stânga, dreapta sau mergând înainte. Există date despre gradul de ocupare al fiecărei intersecții pe fiecare bandă și se cunosc timpii de verde ai fiecărei faze. Se cere să se calculeze ruta unui vehicul care dorește să treacă prin sistemul de intersecții astfel încât timpul total de așteptare la semafoare să fie minim. Este permisă găsirea unei soluții aproximative, bazată pe estimare.
- Considerând sistemul feroviar dat în laboratorul 5 (rețeaua feroviară din Figura 14 și întârzierile trenurilor din Figura 16), se cere să se introducă două trenuri noi *charter*: unul circulă din gara din stânga spre dreapta, celălalt în sens opus. Să se determine rutele și întârzierile acestor trenuri astfel încât să nu existe accidente și timpul total de așteptare să fie minim.

8. Control fuzzy pentru un sistem de intersecții

- Se dau două intersecții în cruce cu o stradă de legătură între ele. Se consideră că la orice rută de intrare într-o intersecție *flow-split*-ul este astfel: 50% dintre vehicule merg înainte, 20% la dreapta și 30% la stânga. Există senzori care furnizează lungimea cozii pe fiecare dintre benzile de intrare în intersecție. Să se stabilească un ciclu al fazelor și să se genereze reguli de control fuzzy pentru fiecare intersecție și

fiecare fază, astfel încât timpul de așteptare la semafor pentru vehicule să fie minim. Se va folosi un simulator existent sau construit special pentru această aplicație.

- Se consideră un sistem de 9 intersecții în cruce, așezate într-o matrice 3×3. Vehiculele pot merge doar înainte. Există senzori care măsoară lungimea cozii de vehicule pe fiecare bandă de intrare, iar semafoarele sunt controlate cu reguli fuzzy de control. Se cere aflarea celor mai bune reguli de control astfel încât timpul total de așteptare la semafor să fie minim.
  Observație: dacă numărul de reguli de control este prea mare, el poate fi redus dacă se consideră *demand*-ul pentru fiecare fază, iar apoi durata fazelor pe baza *demand*-ului.

9. Determinarea controlului ETPN pentru un lac de acumulare

Se dă sistemul de mai jos, format dintr-un lac de acumulare și două generatoare de energie electrică. Există 4 senzori care generează evenimente atunci când nivelul *h* al apei în lac ajunge în dreptul senzorului: *m*, *L*, *H*, *M*. Se cere să se construiască un sistem de control cu evenimente discrete (în forma ETPN) care respectă următoarele specificații:

1. generatorul G_1 funcționează când nivelul apei *h* depășește L
2. generatorul G_1 funcționează când nivelul apei *h* depășește H
3. când nivelul apei depășește M se deschide și golirea c_3.

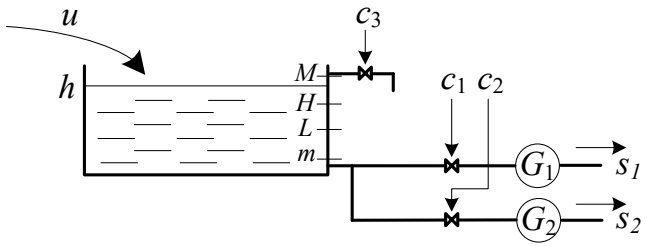


Figura 26. Modelul unui lac pentru control cu ETPN

10. Controlul unei intersecții folosind ETPN

Se consideră o intersecție în cruce cu 3 benzi de intrare din fiecare direcție. Un vehicul care se apropie de intersecție are 50% probabilitate de a se înscrie pe banda de mers înainte, 20% la dreapta și 30% la stânga (*flow-split*). Există senzori pe fiecare dintre benzile de intrare în intersecție, care generează un eveniment atunci când un vehicul se așează la coadă. Să se construiască un sistem de control ETPN care satisface următoarele cerințe: nici un vehicul nu trebuie să aștepte mai mult de 30 de secunde și timpul total de așteptare să fie minim. Se consideră că un vehicul trece prin intersecție în 1 secundă.

11. Optimizarea multiobiectiv a unor funcții matematice

- Se dă funcția $f : [-\pi/2, \pi] \rightarrow R \times R$, $f(x) = (f_1 ; f_2) = (x \cdot \sin(x) ; x \cdot \cos(x))$. Aflați valoarea lui *x* pentru care f_1 și f_2 au valori cât mai mari. Reprezentați grafic frontul Pareto dând valori lui *x* în intervalul $[-\pi/2, \pi]$.

- Se dă funcția f :[-5,5]×[-5,5]→R×R×R, $f(x,y)=\left(x\cdot 1.5^{-x^2-y^2};\sin x\cdot\cos y;\sqrt{x+y}\right)$. Aflați valorile x și y cu o precizie $\delta = 0.01$ pentru care toate componentele funcției f au valori maxime. Reprezentați grafic frontul Pareto ca o suprafață în 3 dimensiuni.

12. Implementarea unui operator de selecție pentru optimizarea multiobiectiv

Pentru problema de optimizare multiobiectiv a unei funcții matematice (prezentată în laboratorul 7), implementați un operator de selecție care să asigure o varietate suficientă în generația următoare astfel: se ia o pondere $\gamma \in [0, 1]$ care scalează cele două obiective, pentru fiecare valoare posibilă a lui γ se calculează performanța ponderată $J_\gamma = \gamma \cdot f_1 + (1- \gamma) \cdot f_2$ și se aleg 2 indivizi cu cea mai mare performanță J_γ. Considerați pentru γ un număr de 100 de valori diferite.

13. Calculul mărimii de control multiobiectiv

- Pentru sistemul descris în laboratorul 3 (studiu de caz I), se adaugă un cost de schimbare a valorii de intrare u_k de la un moment la altul: $c_k = |u_k - u_{k-1}|$. Acest cost are sensul energiei consumate de actuator. Se cere să se afle valorile lui u_k pentru k=0,…,13 astfel încât să se respecte referința y_{ref} dată grafic în Figura 5 și energia totală consumată să fie cât mai mică. Reprezentați grafic frontul Pareto aferent celor 2 obiective.
- Se dau două sisteme de gradul 2, cu o constantă de timp comparabilă τ. Se cere găsirea parametrilor unui controler PI care să regleze cât mai bine răspunsul la treaptă pentru ambele sisteme, pe un orizont de optimizare $T_{sim} = 3\cdot\tau$. Reprezentați grafic frontul Pareto pentru cele două obiective (suma erorilor de reglaj pentru fiecare dintre cele două sisteme).

14. Sistem de control multiobiectiv fuzzy

Se consideră sistemul de 4 lacuri de acumulare din Figura 24. Fiecare lac este controlat independent: se măsoară debitul de intrare și nivelul lacului și se calculează debitul de ieșire folosind un controler fuzzy. Debitul de ieșire este direct proporțional cu energia electrică generată. Există o funcție dată care descrie cererea de energie generată totală: într-o zi există 2 perioade cu cerere mai ridicată (dimineața și seara).

Se cere să se afle regulile de control fuzzy prin optimizare multiobiectiv: nivelul fiecărui lac să fie cât mai aproape de referință și energia totală generată să fie cât mai aproape de cerere în fiecare moment.

15. Determinarea unor expresii matematice / logice

- Se dă o mulțime de numere întregi M = {12, 5, -3, 7, 0, 11, 2} și setul de operații S = {+, -, *, /, ^}. Stabiliți o expresie matematică folosind acești operanzi și cel puțin 4 numere distincte din M, astfel încât rezultatul evaluării expresiei să fie 110.
- Se dă un set de 10 variabile logice (cu valori posibile *true* sau *false*) și o listă de date de test care conțin valorile variabilelor de intrare și o valoare corespunzătoare pentru ieșire. Se cere să se determine o expresie logică cât mai scurtă care generează aceeași

ieșire pentru toate cazurile de test. O expresie logică este formată cu operatorii {*and*, *or*, *not*}

- Se dă un număr natural *N*. Folosind operatorii +, -, *, / (împărțire întreagă), găsiți o expresie matematică ce folosește cât mai puține numere prime ≤ *N*/2, astfel încât rezultatul evaluării expresiei să fie cât mai aproape de *N*.
- Se cere determinarea unei expresii matematice cât mai simple care produce aceleași rezultate cu funcția $F(x)$ pentru $x \in [-10, 10]$:

$$F(x) = \frac{x}{2 \cdot \sin^2(x) + \cos(2x) + tg(x)}$$

16. Probleme de alocare a resurselor

- La o nuntă sunt invitate *N* persoane, iar unele se cunosc între ele în funcție de o matrice de cunoștințe dată. Se cere ca toate persoanele să fie așezate la *m* mese cu câte *L* locuri în așa fel încât la fiecare masă să fie cât mai puține persoane care nu se cunosc între ele. Evident, datele problemei satisfac condiția $N \leq m \cdot L$.
  Bonus: Se pot suplimenta locurile la fiecare masă cu maxim 2 locuri, dar acest lucru este penalizat.
- Într-o companie există *n* angajați, fiecare cu un anumit set de *skill*-uri și un salar dat. Există *p* proiecte, fiecare cu un necesar de număr de oameni cu anumite *skill*-uri. Se cere să se aloce oamenii la proiectele date (nu există alocare part-time), astfel încât proiectele să fie acoperite în totalitate și costurile să fie cât mai mici (oamenii nealocați la proiecte nu intră în cost).

17. Probleme de strategie și joc

- Se cere să se plaseze 8 regine pe o tablă de șah astfel încât ele să atace toate pătratele rămase libere.
- Se dă o tablă ca de șah, dar cu *n*×*m* pătrate, se cere să se determine numărul minim de regine și unde sa fie plasate astfel încât ele să atace toate pătratele ramase libere.
- *Problema sariturii calului*: pe o tablă de șah se află un cal pe pozitia (1,1). Săritura calului este cea obișnuită pe o tablă de șah, în formă de L. Se cere să se determine în ce ordine trebuie să sară ca să acopere toată tabla de joc și să nu calce de 2 ori pe acelasi pătrat.
- Pentru jocul de dame, se cere implementarea unui program pentru care, dacă se dă o configurație a jocului la un anumit moment, oferă cea mai bună variantă de mutare. O variantă de mutare se consideră că este cu atât mai bună cu cât:
  1. se fac mai multe sărituri cu piesa respectivă
  2. se capturează mai multe piese ale adversarului
  3. locul final al piesei este mai aproape de marginea opusă
     Reprezentați frontul Pareto (cele mai bune mutări găsite în funcție de aceste 3 obiective) pe un grafic tridimensional.

Bibliografie

[1] Tiberiu S. Leția, Note de curs Sisteme Evolutive (SE), 2019

[2] Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press. ISBN 9780585030944.

[3] JGAP - Java Genetic Algorithm Package, https://sourceforge.net/projects/jgap/

[4] B.M. Kim, Y.B. Kim, C.H. Oh, A study on the convergence of genetic algorithms, Computers & Industrial Engineering, Volume 33, Issues 3–4, 1997, pag. 581-588, ISSN 0360-8352, https://doi.org/10.1016/S0360-8352(97)00198-8.

[5] Kalyanmoy Deb, An efficient constraint handling method for genetic algorithms, Computer Methods in Applied Mechanics and Engineering, Volume 186, Issues 2–4, 2000, Pages 311-338, ISSN 0045-7825, https://doi.org/10.1016/S0045-7825(99)00389-8.

[6] Peter J. Fleming, Robin C. Purshouse. "Evolutionary algorithms in control systems engineering: a survey." Control engineering practice 10.11 (2002): pag. 1223-1241.

[7] Peter J. Fleming, Carlos M. Fonseca. "Genetic algorithms in control systems engineering." IFAC Proceedings Volumes 26.2 (1993): 605-612.

[8] Cezary Z. Janikow, Zbigniew Michalewicz, "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms", Proceedings of the Fourth International Conference on Genetic Algorithms: 31–36. Retrieved 2 July 2013, http://www.cs.umsl.edu/~janikow/publications/1991/GAbin/text.pdf

[9] Nowaková J., Pokorný M. (2014) "System Identification Using Genetic Algorithms". In: Kömer P., Abraham A., Snášel V. (eds) Proceedings of the Fifth International Conference on Innovations in Bio-Inspired Computing and Applications IBICA 2014. Advances in Intelligent Systems and Computing, vol 303. Springer, Cham. https://doi.org/10.1007/978-3-319-08156-4_41

[10] Tiberiu Leția, Sisteme de control distribuit, note de curs (2019)

[11] Tiberiu Leția., Kilyen Attila, "Fuzzy logic enhanced time Petri net models for hybrid control systems." 2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). IEEE, 2016.

[12] O. Cuibus, T. Letia, "Cooperative control achieved by generic genetic fuzzy logic." Journal of Control Engineering and Applied Informatics 14.3 (2012): 43-53.

[13] R. Făt, O. Cuibus, T. Letia, C. Cenan "Techniques for Assessing Return and Risk of Investment Portfolios: a Case Study." 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR). IEEE, 2020.

[14] Clarke, M., et al. "Allocating railway platforms using a genetic algorithm." Research and Development in Intelligent Systems XXVI. Springer, London, 2010. 421-434.

[15] M.M. Santa, O. Cuibus, T. Letia, 2010. "Train Scheduling with Delay Time Petri Nets and Genetic Algorithms", Proceedings of the 14th International Conference on System Theory and Control (Joint conference of SINTES14, SACCS10, SIMSIS14), 17-19 octombrie, Sinaia, pp. 479-484.

[16] M.M. Santa, O. Cuibus, T. Letia, "Genetic Algorithm for Transitions Scheduling Guidance of Trains", in: Acta Technica Napocensis: Electronica - Telecomunicatii; Cluj-Napoca Vol. 52, Iss. 3, (2011): 14-17.

[17] Vanneschi L., Poli R. (2012) Genetic Programming, "Introduction, Applications, Theory and Open Issues", In: Rozenberg G., Bäck T., Kok J.N. (eds) Handbook of Natural Computing. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-92910-9_24

[18] T. Leția, O. Cuibus. "Automatic linear robot control synthesis using genetic programming." Robot Intelligence Technology and Applications 2. Springer, Cham, 2014. 601-618.

[19] T. Leția, M. Hulea, O. Cuibus. "Controller synthesis method for discrete event systems." Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics. IEEE, 2012.

[20] O. Cuibus, T. Leția. "Genetic programming synthesis of discrete event controllers applied to urban vehicle traffic control." Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics. IEEE, 2012.

[21] A. Konak, D.W. Coit, A.E. Smith, 2006. "Multi-objective optimization using genetic algorithms: A tutorial", Reliability Engineering & System Safety, 91(9), pp.992-1007.

[22] T.S. Leția, O. Cuibus, M. Hulea, R. Miron, Automatic Control Synthesis of Hydro-Power Systems, IFAC Proceedings Volumes, Volume 46, Issue 6, 2013, pages 119-124,ISSN 1474-6670, ISBN 9783902823328, https://doi.org/10.3182/20130522-3-RO-4035.00027.

[23] T. Leția, O. Cuibus., M. Hulea, R. Miron, 2013. Automatic Control Synthesis of Hydro-Power Systems. IFAC Proceedings Volumes, 46(6), pp.119-124.